Université Mohammed Premier
Faculté Pluridisciplinaire
Nador

Polycopié :

Programmation Web

Professeur : *Mohamed ATOUNTI*

Département de Mathématiques et Informatique

Filière : Sciences Mathématiques et Informatique

Semestre : 6

Années universitaire : 2014-2018

Table des matières :

Chapitre 1 : Programmation JavaScript

- Introduction
- Intégrer un script JavaScript à une page Web
- Fonctionnement de base des objets en JavaScript
- Les objets du navigateur
- Création d'objets personnalisés
- Gestionnaires d'événements
- Manipulation d'éléments de page web

Chapitre 2 : XML

- Introduction à XML
- Intérêts
- Syntaxe et structure de XML
- Composition globale d'un document
- Exemples
- XInclude
- DTD
- Espaces de noms
- Scémas XML

Chapitre 3 : PHP

- Intrduction
- Notions essentielles
- Principe de fonctionnement
- Le langage PHP
- Affichage d'une image et du texte
- Les variables

Les Constantes

Les opérateurs

Les structures de contrôles

Les boucles

Les tableaux

Les fonctions

Les fichiers

Programmation modulaires

Création des formulaires

Introdusction aux bases de données

Acces aux SGBD

Connexion au serveur de données

Exploitation des requêtes

Insertion des données.

CHAPITRE 1 :

PROGRAMMATION JAVASCRIPT

1. *Introduction*

JavaScript est un langage de programmation très simple et constitue une excellente initiation à la programmation web.

Il est facile d'utiliser JavaScript pour améliorer les pages Web crées en HTML.

Cependant il est possible d'utiliser JavaScript pour créer des applications complexes.

Un **script** en JavaScript peut être constitué d'une seule ligne ; il peut également être une application complète.

Le script s'exécutera dans un navigateur.

JavaScript, pour sa part est un langage interprété : le navigateur exécute chacune des lignes du programme au fur et à mesure.

Il est aussi simple de changer le contenu d'un script JavaScript que celui d'un document HTML.

2. *Intégrer un script JavaScript à une page Web*

<script> : interpréter la suite du document comme un script.

</script> : interpréter la suite du document comme un HTML.

Les instructions JavaScript doivent toujours être placées à l'intérieur des balises **<script></script>**.

Un script peut être placé à quatre endroits différents :

- Dans le corps de la page : le résultat du script s'affiche au sein du document HTML lorsque la page est chargée.
- Dans l'en-tête de la page, à l'intérieur des balises Head : lorsqu'un script est placé dans l'en-tête, il n'est pas exécuté immédiatement, mais d'autres scripts peuvent s'y référer. L'entête est souvent employé pour les fonctions.
- A l'intérieur d'une balise HTML : on appelle cela un gestionnaire d'événements ; il permet au script d'interagir avec des éléments HTML.

- Dans un fichier séparé : vous pouvez stocker des scripts dans des fichiers comportant l'extension .js ; pour vous y référer vous indiquerez le nom de fichier dans la balise <script>.

- Une procédure est une suite d'instructions qui forment un tout et qui sont regroupées sous un même nom.
- Une fonction est une suite d'instructions qui calcule un résultat; celui-ci est transmis à l'expression qui a appelé la fonction, après le mot return.
- De plus, procédures et fonctions peuvent admettre des paramètres.
- S'il n'y a pas besoin de paramètres, le nom de la fonction est suivi d'un couple de parenthèses vides.

3. *Fonctionnement de base des objets en JavaScript*

- Un ***objet*** JavaScript est constitué de ses ***propriétés*** (sa description, ses caractéristiques) et de ***méthodes*** (ce qu'il sait faire).

  Les ***objets*** permettent de combiner différents types de données (propriétés) et de fonctions permettant d'agir sur ces données (méthodes) en un seul élément facile à manipuler.

- La ***propriété*** d'un objet est l'équivalent d'une variable qui serait contenue dans un objet.

  Syntaxe : ***objet.propriété***

- Les ***méthodes*** sont des fonctions stockées en tant que propriétés d'objets.

  Syntaxe : ***objet.méthode(argument)***

- Le mot clé ***with*** permet de simplifier l'écriture des programmes en JavaScript ou du moins la quantité de code à saisir.

  Le mot clé ***with*** permet de spécifier un objet : il est suivi d'un bloc d'instructions placées entre accolades. Pour chacune des instructions du bloc, les propriétés mentionnées sans que l'objet correspondant soit indiqué se réfèrent à l'objet indiqué après ***with***.

- **L'objet Math**

  Math est un objet prédéfini de JavaScript qui comprend de nombreuses constantes et fonctions.
  Les propriétés de l'objet ***Math*** sont des constantes mathématiques, ses méthodes des fonctions mathématiques.

  Liste des principales méthodes

  - Math.sqrt() , racine carrée.
  - Math.log() , Math.exp() , Math.abs() ,Math.cos (), Math.sin() , Math.tan()
  - Math.floor(), Math.ceil(),entier immédiatement inférieur /supérieur.
  - Math.pow(base, exposant), fonction puissance, où base et exposant sont des expressions numériques quelconques évaluables.
  - Math.max() , Math.min()
  - Math.random(), nombre "réel" choisi au hasard dans [0 , 1[.
  - Math.round(), arrondit à l'entier le plus proche.

- **Les dates**

  Date est un objet prédéfini de JavaScript qui permet de manipuler facilement dates et heures.

  L'origine des dates a été choisie le 1er janvier 1900 et est exprimée en millisecondes.

  Pour construire un objet de type **Date**, il faut utiliser un constructeur Date() avec le mot-clé new

  variable = **new Date**(*liste de paramètres*)

  Les méthodes ***set*** permettent définir les valeurs des objets Date.

  setDate() : définit le jour du mois ;

  setMonth() : définit le mois;

  setFullYear() : définit l'année en quatre chiffres ;

  setHours(), setMinutes() et setSecondes() : définit l'heure.

Les méthodes ***get*** permettent de lire les valeurs des objets Date :

getDate() : permet de lire le jour du mois ;
getMonth() : permet de lire le mois ;
getFullYear() : permet de lire l'année;
getHours(), getMinutes() et getSecondes() permettent de lire les heures, les minutes et les secondes.

4.Les objets du navigateur

- **Le fonctionnement des objets de navigateur**

L'un des avantages de JavaScript est qu'il permet de manipuler directement le navigateur Web.

Vous pouvez utiliser un script JavaScript pour charger une nouvelle page dans le navigateur, manipuler des parties de la fenêtre et même ouvrir de nouvelles fenêtres.

Pour effectuer ces manipulations, JavaScript fait appel à des objets de navigateur.

Les objets du navigateur ont des propriétés qui décrivent la page Web ou le document et des méthodes qui permettent de les modifier.

L'objet ***window*** se situe au sommet de la hiérarchie des objets de navigateur.

- **Manipulation des documents Web**

L'objet ***document*** représente un document Web.

Les documents Web sont affichés dans la fenêtre du navigateur

Dans la mesure où l'objet ***window*** représente toujours la fenêtre en cours (celle qui contient le script), vous pouvez utiliser ***window.document*** pour vous référer au document en cours.

Vous pouvez également vous référer uniquement à ***document*** : vous désignez ainsi automatiquement le document de la fenêtre en cours.

Si plusieurs fenêtres ou cadres sont ouverts, il existera plusieurs objets ***window*** et un objet ***document*** pour chacun d'eux.

Propriétés d'un objet ***document :***

location : indique l'URL du document.

title : représente le titre du document.

referrer : l'adresse d'où l'on vient.

lastModified : est la date de dernière modification de la page.

linkcolor : la couleur des liens.

Quelques **méthodes** de l'objet **document** :

write() : écrit du texte ou de l'html dans le document.
writeln() : idem, mais suivi d'un retour-chariot.
close() : fermeture du document en cours.

- ## Accès à l'historique du navigateur

L'objet ***history*** est une propriété fille de l'objet ***window***.

Il contient des informations sur les URL qui ont été visitées avant et après la page courante, ainsi que des méthodes pour y accéder :

L'objet ***history*** dispose de trois méthodes :

- ***history.go()*** : qui ouvre une URL de l'historique. Pour l'utiliser, indiquez un nombre positif ou négatif comme paramètre.
- ***history.back()*** : qui affiche la page précédente.
- ***history.forward()*** : qui affiche la page suivante.

L'objet ***history*** dispose de quatre propriétés :

- ***history.length*** : indique la longueur de l'historique; ie : le nombre de pages différentes visitées par l'utilisateur.
- ***history.current*** : qui contient la valeur courante de l'historique, l'URL de la page que l'utilisateur est en train de visualiser.
- ***history.next*** : cette propriété ne contient pas de valeur que si l'utilisateur à déjà cliqué sur Précédente.
- ***history.previous*** : qui est la valeur de l'élément précèdent de l'historique, ie : la page où sera envoyé l'utilisateur s'il clique sur précédente.

- **L'objet *location***

  L'objet location est lui même fille de l'objet window. Il stocke des informations concernant l'URL de la page affichée dans la fenêtre et permet de charger de nouvelles pages.

  L'objet location dispose de deux méthodes :

- ***location.reload()*** : qui permet de recharger le document courant et qui équivaut à un clic sur actualiser.

- ***location.replace()*** : qui remplace la page courante par une nouvelle page. Cette méthode est similaire à l'emploi de location.href pour charger une nouvelle page, à la différence qu'avec location.replace(), l'historique du navigateur n'est pas modifié.

  Autrement dit, l'utilisateur ne pourra pas revenir à la page précédemment affiché à l'aide de Précédente.

5. *Création d'objets personnalisés*

- **Définir un objet**

  La première étape de la création d'un objet consiste à lui donner un nom et à nommer ses propriétés.

  Puis on doit créer une fonction *(ce qu'on avait appelé : Constructeur)* pour pouvoir instancier notre objet.

  Le constructeur est une fonction simple acceptant des paramètres pour initialiser le nouvel objet, puis les affectant aux propriétés correspondantes.

- Définir une méthode pour un objet :On créera une méthode pour manipuler l'objet**,** puis on créera une méthode qui sert à afficher les propriétés de l'objet.
- Création d'instances d'objet : Pour créer une instance de l'objet défini, on utilise le mot clé ***new***.

6. ***Gestionnaires d'événements***

L'utilisateur déclenche un "événement" (clic, déplacement souris, clic sur un bouton, choix d'un option de liste déroulante etc ...) relativement à un objet (lien, composant de formulaire ..).

L'événement est décelé (capté) par l'objet cible si celui-ci possède une "sensibilité" à l'événement. Il faut donc connaitre la correspondance objet-événement.

S'il prévoit un intérêt à "répondre" à cet événement, le programmeur associera du code JS ou une fonction JS à un tel couple objet-événement.

L'appel et l'exécution de ce code ou de cette fonction seront automatiquement déclenchés par l'événement, et constituent ainsi la "réponse" à celui-ci.

Les scripts qui font appel à des gestionnaires d'événements interagissent directement avec l'utilisateur au lieu de s'exécuter de manière ordonnée.

Exemple :

<A href=www.google.com onMouseOver="window.alert('vous êtes passé au dessus du lien menant vers google ');"> Cliquer ici </A>

/* une boite de dialogue s'affiche lorsque la souris passe au-dessus du lien */

Vous pouvez utiliser directement des instructions JavaScript en tant que gestionnaires d'événements, mais on peut aussi faire appel à une fonction.

<A href=www.google.com onMouseOver="action();"> Cliquer ici </A> /* il faut définir au préalable la fonction action() */

- Utilisation de l'objet ***event***

  L'objet ***event*** est un objet spécial envoyé à un gestionnaire d'événement chaque fois qu'un événement se produit.

  Les propriétés de l'objet ***event*** fournissent des informations supplémentaires sur l'événement qui s'est produit.

  type : c'est le type d'événement qui s'est produit ; Exp : mouseover.
  target : c'est l'objet cible de l'événement ; Exp : lien, bouton, ….

which : est une valeur numérique indiquant quel bouton de souris a été cliqué (pour les événements de souris) ou quelle touche a été enfoncée (pour les événements de clavier).

modifiers : est la liste des touches de modification (Alt, Ctrl, Maj…) enfoncées lors d'un événement de souris ou de clavier.

pageX et pageY : sont les positions X et Y de la souris au moment où l'événement s'est produit.

- Emploi des événements de souris

  JavaScript pourra détecter les déplacements du pointeur de souris et le fait que l'un ou l'autre de ses boutons soit cliqué, enfoncé ou relâché.

  - ***onMouseOver*** : ce gestionnaire est appelé lorsque le pointeur de la souris se trouve au dessus d'un lien, d'une image ou d'un autre objet.
  - ***onMouseOut*** : celui-là fonctionne de manière opposée au premier, il est appelé lorsque le pointeur de la souris quitte l'objet.
  - ***onClick*** : celui-là est appelé lorsqu'on a cliqué une seule fois sur un des boutons de la souris.
  - ***onMouseDown*** : lorsque l'utilisateur enfonce le bouton de la souris.
  - ***onMouseUp*** : lorsque l'utilisateur relâche le bouton de la souris.

  Pour savoir lequel des boutons de la souris ont été enfoncé, vous pouvez utiliser la propriété ***which*** de l'objet ***event***.

  Cette propriété contient la valeur 1 si c'est le bouton gauche et 3 si c'est le bouton droit.

- L'événement ***onload***

  Cet événement se produit lorsque le téléchargement de la page en cours est fini avec toutes ses images.

  Pour définir ce gestionnaire, on l'appelle dans la balise ***body***.

  Exemple:

  <body onload="alert('chargement de page terminé');">

7. ***Manipulation d'éléments de page Web***

- **L'objet *window***

  L'objet ***window*** se réfère toujours à la fenêtre courante ie : celle qui contient le script.

  Plusieurs fenêtres peuvent être ouvertes simultanément, et on utilisera leurs noms respectifs pour vous référer à chacune d'elles.

- **Créer une nouvelle fenêtre**

  L'une des utilisations les plus intéressantes de l'objet ***window*** est la création de nouvelles fenêtres.

  On peut ainsi afficher un nouveau document sans effacer l'ancien.

  Pour créer une nouvelle fenêtre, on utilise la méthode ***window.open().***

  Objetfenetre = window.open("URL","Nom de la fenêtre","Liste des caractéristiques");

  Objetfenetre : cette variable permet de stocker le nouvel objet ***window***. Pour accéder aux propriétés et aux méthodes de cet objet, on utilisera le nom de cette variable.

  URL : en l'occurrence celle du document qui sera affiché dans la nouvelle fenêtre. Si ce paramètre n'est pas mentionné, aucune page ne sera affichée.

  Nom de la fenêtre : ce paramètre indique le nom de la fenêtre. Ce nom sera affecté à la propriété ***name*** de l'objet ***window***.

  Le troisième paramètre est une liste de caractéristiques optionnelles, séparées par des virgules. Il permet de personnaliser la nouvelle fenêtre en choisissant d'y inclure la barre d'outils, la ligne d'état et d'autres éléments.

  Détails :

  les caractéristiques du troisième paramètre de la méthode window.open() sont, entre autres, *width* et *height* pour déterminer la taille de la fenêtre ; ainsi qu'une série de caractéristiques pouvant être

activées (1 ou yes) ou désactivées (0 ou no) : *toolbar*, *directories*, *status*, *menubar*, *scrollbars* et *resizable*. Si vous omettez l'une de ces caractéristiques, c'est la valeur par défaut qui sera utilisée.

- **Fermeture de fenêtres**
  La méthode ***window.close()*** permet de fermer les fenêtres du navigateur.

  Exemple:

  Petitefenetre.close(); // cette instruction permet de fermer la fenêtre Petitefenetre

- Insertion de pauses dans l'exécution d'un script

  JavaScript possède une fonction prédéfinie en sorte qu'un script ne fasse rien, et ce pour une période déterminée.

  La méthode ***window.setTimeout()*** permet de spécifier un délai et une commande, laquelle s'exécutera une fois le délai écoulé.

  La méthode ***setTimeout()*** comprend deux paramètres : le premier est une instruction (un groupe d'instructions) en JavaScript entre guillemets et le second est un délai en *ms.*

  Exemple :

  Id = window.setTimeout("alert(' Les 20 secondes se sont écoulés')",20000);

  Une variable (ici : Id) permet de stocker un identificateur ; vous pouvez ainsi créer plusieurs pauses, chacune ayant un identificateur particulier.

  Vous pouvez ensuite interrompre la pause à l'aide de la méthode ***clearTimeout()*** en spécifiant son identificateur :

  Exemple :

  Window.clearTimeout(Id);

- Les cadres et JavaScript

  La plupart des navigateurs actuels prennent en charge les cadres ou ***frames***, qui permettent de diviser la fenêtre du navigateur en plusieurs

volets. Chaque volet peut afficher un document HTML séparé.

- Objets JavaScript permettant de manipuler les cadres :

  Lorsqu'une fenêtre contient plusieurs cadres, chacun d'eux est représenté en JavaScript par un objet ***frame***.

  Cet objet est équivalent à un objet ***window***, à cela prêt qu'il sert de manipuler les cadres. Le nom de l'objet ***frame*** est celui qu'on lui donne avec l'attribut ***name*** à l'intérieur de la balise <*FRAME*>.

- **Le tableau frames**

  Au lieu de vous référer aux cadres d'un document par leurs noms, utilisez le tableau ***frames***.

  Ce tableau stocke les informations concernant chacun des cadres d'un document.

  Le numéro d'indice des cadres commence à 0 et avec la première balise <frame> du document frameset.

- **Les formulaires**

  JavaScript permet et rend les formulaires HTML plus interactifs, de valider les informations saisies par l'utilisateur et d'afficher les données en fonction d'autres données.

- Les formulaires HTML : JavaScript permet d'ajouter diverses fonctions très utiles.

Les formulaires HTML commencent par la balise <*FORM*> et se terminent par <*/FRAME*>.

La balise **<*FORM*>** comprend trois paramètres :

NAME : qui est le nom du formulaire.

METHOD : qui peut avoir les valeurs GET et POST ; il s'agit des deux méthodes d'envoi des réponses au serveur.

ACTION : qui est le script CGI auquel les réponses seront envoyées.

Exemple :

<FORM NAME="commandes" METHOD="GET" ACTION="commandes.cgi">

L'exemple ci-dessus de la balise <FORM> est un formulaire nommé commandes. Ce formulaire utilise la méthode GET et envoie les données recueillies à un script CGI nommé commandes.cgi qui se trouve dans le même répertoire que la page web.

- Utilisation de l'objet *form*

Chaque formulaire d'une page Web est représenté en JavaScript par un objet ***form***, dont le nom est identique à l'attribut NAME de la balise <FORM> qui a contribué à sa création.

On peut aussi utiliser le tableau ***forms*** pour se référer aux formulaires d'une page.

Ce tableau regroupe tous les formulaires de la page, le premier recevant l'indice 0.

Si le premier formulaire du document s'appelle ***formulaire1***, on a le choix de se référer à ce formulaire comme suit :

document.formulaire1
document.forms[0]

- Propriétés de l'objet ***form***

action : correspond à l'attribut ACTION du formulaire ; soit le programme auquel les réponses du formulaire seront soumises.

length : c'est le nombre d'éléments du formulaire. Cette propriété ne peut être modifiée.

method : c'est la méthode employée pour envoyer le formulaire, GET ou POST.

target : c'est la fenêtre dans laquelle le résultat du formulaire s'affichera. En principe, on utilise la fenêtre principale, et c'est le formulaire lui même qui est remplacé.

- Envoyer et réinitialiser des formulaires

  L'objet ***form*** dispose de deux méthodes, ***submit*** et ***reset***.

  On peut utiliser ces méthodes pour envoyer les données ou réinitialiser le contenu du formulaire sans que l'utilisateur ait besoin de cliquer sur un bouton.

  Ces méthodes pourront vous être utiles dans le cas où l'utilisateur clique sur une image, un lien ou effectue une autre action qui, en principe, ne permet pas d'envoyer les réponses.

  - Détecter les événements de formulaire

    L'objet ***form*** dispose de deux gestionnaires d'événements, ***onSubmit*** et ***onReset***.

    On peut spécifier un ensemble d'instructions JavaScript ou un appel de fonction pour ces événements en les plaçant dans la balise ***<FORM>*** qui définit le formulaire.

- Eléments de formulaire et JavaScript

  La propriété importante de l'objet ***form*** est le tableau ***éléments***, qui contient un objet pour chacun des éléments du formulaire.

  On peut se référer à un élément par son nom ou son numéro d'indice dans le tableau.

  document.commandes.elements[0]
  document.commandes.nom1

  Si notre premier formulaire; alors on utilise l'expression suivante :

  document.forms[0].éléments[0]
  document.forms[0].nom1

  Généralement on utilise les noms des formulaires et des éléments ; c'est la manière la plus simple de s'y référer.

  document.forms.length : renvoie le nombre de formulaires du document.
  document.forms[0].elements.length : renvoie le nombre d'éléments du premier formulaire du document.

Champs de texte : l'élément de formulaire le plus couramment employé est le champ de texte.

Celui-ci servira à demander à l'utilisateur tout genre d'information.
JavaScript permet d'afficher automatiquement du texte dans un champ de texte.

Exemple :

<INPUT TYPE="TEXT" NAME="texte1" VALUE="bonjour" SIZE="30">

On définit ainsi un champ de texte nommé *texte1*. Le champ a la valeur par défaut *"bonjour"* et permet la saisie d'un maximum de 30 caractères.

JavaScript réfère à ce champ comme un objet ***text*** ayant le nom ***texte1***.

Les champs de texte sont les plus faciles à manipuler à l'aide de JavaScript.

Chaque objet ***text*** dispose des propriétés suivantes :

name : qui est le nom du champ. Il est aussi utilisé comme nom de l'objet.

defaultvalue : qui est la valeur par défaut du champ. Elle correspond à l'attribut *VALUE*.

Il s'agit d'une propriété en lecture seule.

value : qui est la valeur courante. Initialement, elle est identique à la valeur par défaut, mais elle peut être modifiée par l'utilisateur ou par une fonction JavaScript.

En général, la manipulation de champs de texte consiste à utiliser l'attribut ***value*** pour lire la valeur saisie par l'utilisateur ou pour modifier son contenu.

Ainsi, l'instruction ci-dessous donne la valeur "champ de texte" au champ de texte type_element du formulaire formulaire_1 :

Exemple :

document.formulaire_1.type_element.value = "champ de texte";

Les zones de texte sont définies à l'aide de la balise <*TEXTAREA*>, et sont représentées par l'objet *textarea*.

Exemple :

<TEXTAREA NAME="texte2" ROWS="2" COLS="70">

Cela est le contenu de la balise TEXTAREA

</TEXTAREA>

- Manipulation du texte de formulaires :

  Les objets ***text*** et ***textarea*** disposent de plusieurs méthodes :

  focus() : pour activer le champ correspondant. Le pointeur vient se placer dans ce champ et celui-ci devient le champ courant.

  blur() : produit l'effet inverse ; il "désactive" le champ.

  select() : sélectionne le texte du champ.

  Les objets ***text*** et ***textarea*** prennent en charge les gestionnaires d'événements suivants :

  onFocus : se produit lorsque le champ de texte est activé.

  onBlur : se produit lorsque le champ de texte est désactivé.

  onChange : se produit lorsque l'utilisateur modifie du texte, puis "sort" du champ en passant à un autre.

  onSelect : se produit lorsque l'utilisateur sélectionne une partie ou tout le texte d'un champ.

Tableau récapitulatif

Gest. événement	provoqué par l'utilisateur qui ...	sur les objets ...
onBlur	enlève le focus du composant	text, textarea, select
onChange	change la valeur d'un texte ou d'un composant à options	text, textarea, select
onClick	clique sur un composant ou un hyperlien	button, checkbox, radio, reset, submit
onFocus	donne le focus au composant	text, textarea, select
onLoad	charge la page dans le navigateur	balises BODY, FRAMESET
onMouseOut	la souris quitte un lien ou une ancre	balises <A HREF..>, <AREA HREF..>
onMouseOver	bouge la souris sur un lien ou une ancre	balises <A HREF..>, <AREA HREF..>
onReset	efface les saisies d'un formulaire	bouton reset
onSelect	sélectionne une zone d'édition d'un formulaire	text, textarea
onSubmit	soumet un formulaire	bouton submit
onUnload	quitte la page	balises BODY, FRAMESET

Chapitre 2 :

XML

1. ***Introduction à XML***

Le langage XML (eXtended Markup Language) est un langage de format de document.

Il dérive de SGML (Standard Generalized Markup Language) et HTML (HyperText Markup Language). Comme ces derniers, il s'agit d'un langage formé de *balises* qui permet de structurer les documents.

Le langage XML s'est imposé comme le format standard pour les communications entre applications.

Il est utilisé dans la plupart des projets de publication sur le WEB ainsi que dans les bases de données.

2. ***Intérêts***

XML s'est imposé comme un standard incontournable dans le monde de l'informatique. Il est aussi bien utilisé pour stocker des documents que pour des échanges de données.

Ce succès est en grande partie du aux qualités de XML.

Nous allons énumérer ces caractéristiques essentielles qui ont conduit à ce développement puis nous allons les détailler plus en profondeur.

- Simplicité, universalité et extensibilité.
- Format texte avec gestion des caractères spéciaux.
- Structuration forte.
- Séparation stricte entre contenu et présentation.
- Modèles de documents (DTD et XML-Schémas).
- Modularité des modèles.
- Validation du document par rapport au modèle.
- Format libre
- Nombreuses technologies développées autour de XML.

Une des idées directrices de XML est la séparation entre contenu et présentation.

Un des principes de XML est d'organiser le contenu de manière indépendante de la présentation.

La séparation est encore plus marquée en XML car la signification des balises n'est pas figée comme en HTML.

La règle est alors de choisir les balises pour organiser le document en privilégiant la structure de celui-ci par rapport à une éventuelle présentation.

Un second principe de XML est une structuration forte du document.

Une des caractéristiques essentielles de XML est son extensibilité et sa flexibilité.

Un ensemble de règles portant sur les documents XML sont appelées ***modèles de documents***. Plusieurs langages ont été développés pour décrire ces règles.

L'intérêt principal de ces modèles de documents est de pouvoir décrire explicitement les règles à respecter pour un document et de pouvoir vérifier si un document donné les respecte effectivement.

Les modèles de document tels que les DTD (*Document Type Description*) ou les schémas peuvent servir à une vérification automatique des documents. Il existe plusieurs implémentations de ces modèles.

Les données présentes dans un document XML soient fortement structurées, le format XML est un format basé sur le texte.

Il est ainsi possible de manipuler un document XML à l'aide d'un simple éditeur de texte. Il n'est pas nécessaire d'utiliser un logiciel spécialisé.

Il existe bien sûr des logiciels spécialement conçus pour l'édition de documents XML.

3. ***Syntaxe et structure de XML***

La syntaxe de XML est relativement simple.

Elle est constituée de quelques règles pour l'écriture d’un **entête** et des **balises** pour structurer les données.

Ces règles sont très similaires à celles du langage HTML utilisé pour les pages WEB mais elles sont en même temps plus générales et plus strictes.

Elles sont plus générales car les noms des balises sont **libres**.

Elles sont aussi plus strictes car elles imposent qu'à toute **balise ouvrante** corresponde une **balise fermante**.

3.1 Premier exemple

Le langage XML est un format orienté texte.

Un document XML est simplement une suite de caractères respectant quelques règles.

Il peut être stocké dans un fichier et/ou manipulé par des logiciels en utilisant un codage des caractères.

Ce codage précise comment traduire chaque caractère en une suite d'octets réellement stockés ou manipulés.

On commence par donner un premier exemple de document XML comme il peut être écrit dans un fichier bibliography.xml.

Ce document représente une bibliographie de livres sur XML.

Ce document contient une liste de livres avec pour chaque livre, le titre, l'auteur, l'éditeur (publisher en anglais), l'année de parution, le numéro ISBN et éventuellement une URL.

<?xml version="1.0" encoding="iso-8859-1"?>[1]
<!- -Time-stamp: "bibliography.xml 3 Mar 2008 16:24:04" - - >[2]
<!DOCTYPE bibliography SYSTEM "bibliography.dtd" >[3]
<bibliography>[4]
<book key="Michard01" lang="fr">[5]
<title>XML langage et applications</title>
<author>Alain Michard</author>
<year>2001</year>
<publisher>Eyrolles</publisher>
<isbn>2-212-09206-7</isbn>
<url>http://www.editions-eyrolles/livres/michard/ </url>
</book>
<book key="Zeldman03" lang="en">
<title>Designing with web standards</title>
<author>Jeffrey Zeldman</author>
<year>2003</year>

<publisher>New Riders</publisher>
<isbn>0-7357-1201-8</isbn>
</book>
…
</bibliography>[6]

[1] Entête XML avec la version 1.0 et l'encodage iso-8859-1 des caractères.

[2] Commentaire délimité par les chaînes de caractères < ! - - et - - >.

[3] Déclaration de DTD externe dans le fichier bibliography.dtd.

[4] Balise ouvrante de **l'élément racine** bibliography

[5] Balise ouvrante de l'élément book avec deux attributs de noms **key** et **lang** et de valeurs Michard01 et fr

[6] Balise fermante de l'élément racine bibliography

3.2 Syntaxe et structure

Pour qu'un document XML soit correct, il doit d'abord être ***bien formé*** et, ensuite, être ***valide***.

La première contrainte est de nature ***syntaxique***. Un document bien formé doit respecter certaines règles syntaxiques propres à XML.

La seconde contrainte est de nature ***structurelle***. Un document valide doit respecter un ***modèle de document***.

Un tel modèle décrit de manière rigoureuse comment doit être organisé le document.

Pour chaque application, il est possible de choisir la grammaire la plus appropriée. Cette possibilité d'adapter la grammaire aux données confère une grande souplesse à XML.

Les DTD (*Document Type Description*), héritées de SGML, sont simples mais aussi assez limitées.

Un document XML est généralement contenu dans un fichier texte dont l'extension est **.xml**. Il peut aussi être réparti en plusieurs fichiers.

Les extensions pour les schémas XML, les feuilles de style XSLT, les dessins en SVG sont par exemple .xsd, .xsl et .svg.

Un document XML est, la plupart du temps, stocké dans un fichier mais il peut aussi être dématérialisé et exister indépendamment de tout fichier.

Une chaîne de traitement de documents XML peut produire des documents intermédiaires qui sont détruits à la fin. Ces documents existent uniquement pendant le traitement et sont jamais mis dans un fichier.

4. ***Composition globale d'un document***

Un document XML est composé de trois constituants suivants.

Prologue : Il contient des déclarations facultatives.

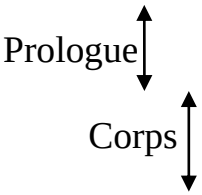
Corps du document : C'est le contenu même du document.

Commentaires et instructions de traitement : Ceux-ci peuvent apparaître partout dans le document, dans le prologue et le corps.

Le document se découpe en fait en deux parties consécutives qui sont le ***prologue*** et le ***corps***. ***Les commentaires et les instructions de traitement*** sont ensuite librement insérés avant, après et à l'intérieur du prologue et du corps.

La structure globale d'un document XML est la suivante.

<?xml ... ?>
...
<*root-element*>
...
</*root-element*>



Dans l'exemple précédent :

Le prologue comprend les trois premières lignes du fichier.

La première ligne est **l'entête** XML et la deuxième est simplement un **commentaire** utilisé pour mémoriser le nom du fichier et sa date de dernière modification. La troisième ligne est la déclaration d'une **DTD** externe contenue dans le fichier bibliography.dtd.

Le corps du document commence à la quatrième ligne du fichier avec la balise ouvrante **<bibliography>**. Il se termine à la dernière ligne de celui-ci avec la balise fermante **</bibliography>**.

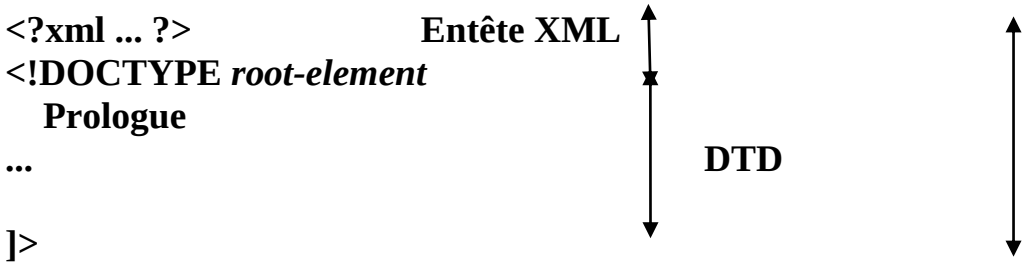
4.1 Prologue

Le prologue contient deux déclarations facultatives mais fortement conseillées ainsi que des commentaires et des instructions de traitement.

La première déclaration est **l'entête XML** qui précise entre autre la version de XML et le codage du fichier.

La seconde déclaration est la déclaration du type du document **(DTD)** qui définit la structure du document. La déclaration de type de document est omise lorsqu'on utilise des **schémas XML** ou d'autres types de modèles qui remplacent les **DTD**.

La structure globale du prologue est la suivante.



- ***Entête XML***

  L'entête utilise une syntaxe <?xml ... ?> .

  L'entête XML a la forme générale suivante.

  <?xml version="..." encoding="..." standalone="..." ?>

  L'entête doit se trouver au début du document. Ceci signifie que les trois caractères '<?x' doivent être les trois premiers caractères du document.

  Cet entête peut contenir trois attributs **version**, **encoding** et **standalone**.

Chaque attribut a une valeur délimitée par une paire d'apostrophes " ou une paire de guillemets "".

L'attribut **version** précise la version d'XML utilisée.

Les valeurs possibles actuellement sont 1.0 ou 1.1.

L'attribut **encoding** précise le codage des caractères utilisé dans le fichier.

Les principales valeurs possibles sont US-ASCII, ISO-8859-1, UTF-8, et UTF-16. Ces noms de codage peuvent aussi être écrits en minuscule comme utf-8.

L'attribut **standalone** précise si le fichier est autonome, c'est-à-dire s'il requiert ou non des ressources extérieures.

La valeur de cet attribut peut être **yes** ou **no**.

L'attribut **version** est obligatoire et l'attribut **encoding** l'est aussi dès que le codage des caractères n'est pas le codage par défaut UTF-8.

Voici quelques exemples d'entête XML.

<?xml version="1.0"?>
<?xml version='1.0' encoding='UTF-8' ?>
<?xml version="1.1" encoding="iso-8859-1" standalone="no" ?>

- ***Déclaration de type de document***

  La déclaration de type définit la structure du document.

  Elle précise en particulier quels éléments peuvent contenir chacun des éléments.

  Cette déclaration de type peut prendre plusieurs formes suivant la définition du type incluse dans le document ou externe.

  Elle a la forme générale suivante qui utilise le mot clé **DOCTYPE** et

  < ! **DOCTYPE** >.

4.2 Corps du document

Le corps du document est constitué de son contenu qui est organisé de façon hiérarchique à la manière d'un système de fichiers à l'exception qu'aucune distinction n'est faite entre fichiers et répertoire.

L'unité de cette organisation est **l'*élément***. Chaque élément peut contenir du texte simple, d'autres éléments ou encore un mélange des deux.

Comme dans une arborescence de fichiers, il y a un élément appelé ***élément racine*** qui contient l'ensemble du document.

- ***Éléments***

Un ***élément*** est formé d'une balise ouvrante, d'un contenu et de la balise fermante correspondante.

La *balise* ***ouvrante*** prend la forme <***name***> formée du caractère '<', du nom ***name*** de l'élément et du caractère '>'.

Des attributs peuvent éventuellement être ajoutés entre le **nom** et le caractère '>'.

La balise ***fermante*** prend la forme **</*name*>** formée des deux caractères '</', du nom ***name*** de l'élément et du caractère '>'.

Les noms des éléments sont des noms XML **quelconques**. Ils ne sont pas limités à un ensemble fixé de noms prédéfinis comme en HTML.

Le ***contenu*** d'un élément est formé de tout ce qui se trouve entre la balise ouvrante et la balise fermante. Il peut être constitué de texte, d'autres éléments, de commentaires et d'instructions de traitement.

Dans la balise ouvrante, le caractère '<' doit être immédiatement suivi du nom de l'élément. En revanche, il peut y avoir des espaces entre le nom et le caractère '>'. La balise fermante ne peut pas contenir d'espace.

<name></name> ou <name/>

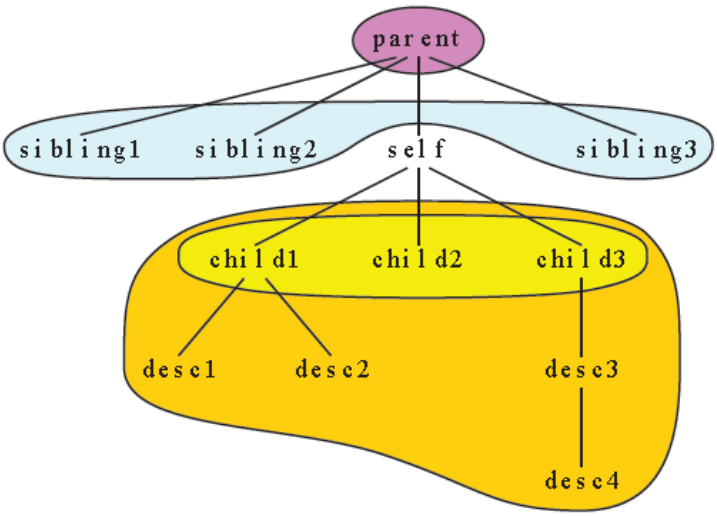
Contenu vide

Elément avec un contenu vide

Lorsque le contenu est vide, c'est-à-dire lorsque la balise fermante suit immédiatement la balise ouvrante, les deux balises peuvent éventuellement se contracter en une seule balise de la forme **<*name*/>** formée du caractère '<', du nom ***name*** et des deux caractères '/>'.

Cette contraction est à privilégier lorsque l'élément est déclaré vide par une DTD.

```
<parent>
   <sibling1> ... </sibling1>
   <sibling2> ... </sibling2>
   <self>
         <child1> ... <desc1></desc1> ... <desc2></desc2> ... </child1>
         <child2> ... </child2>
         <child3> ... <desc3><desc4> ... </desc4></desc3> ... </child3>
   </self>
<sibling3> ... </sibling3>
</parent>
```



Liens de parenté

Dans l'exemple ci-dessus, le contenu de l'élément **self** s'étend de la balise ouvrante <**child1**> jusqu'à la balise fermante </**child3**>. Ce contenu comprend tous les éléments **child1**, **child2** et **child3** ainsi que les éléments **desc1**, **desc2**, **desc3** et **desc4**. Tous les éléments qu'il contient sont appelés ***descendants*** de l'élément **self**.

Parmi ces descendants, les éléments child1, child2 et child3 qui sont directement inclus dans self sans élément intermédiaire sont appelés les ***enfants*** de l'élément **self**.

Inversement, l'élément **parent** qui contient directement self est appelé le ***parent*** de l'élément self. Les autres éléments qui contiennent l'élément ***self*** sont appelés les ***ancêtres*** de l'élément self.

Les autres enfants sibling1, sibling2 et sibling3 de l'élément parent sont appelés les ***frères*** de l'élément self.

Ces relations de parenté entre les éléments peuvent être visualisées comme un arbre généalogique.

- ***Sections littérales***

Les caractères spéciaux '<', '>' et '&' ne peuvent pas être inclus directement dans le contenu d'un document.

Ils peuvent être inclus par l'intermédiaire des entités prédéfinies.

Les *sections littérales*, appelées aussi *sections CDATA (*Character Data) en raison de la syntaxe permettent d'inclure du texte qui est qui recopié**.**

Une section littérale commence par la chaîne de caractères '<![CDATA[' et se termine par la chaîne ']]>'.

Tous les caractères qui se trouvent entre ces deux chaînes font partie du contenu du document, y compris les caractères spéciaux.

<![CDATA[Contenu avec des caractères spéciaux <, > et &]]>

Une section CDATA ne peut pas contenir la chaîne de caractères ']]>' qui permet à l'analyseur lexical de détecter la fin de la section. Il est en particulier impossible d'imbriquer des sections CDATA.

- ***Attributs***

  Les balises ouvrantes peuvent contenir des ***attributs*** associés à des valeurs.

  L'association de la valeur à l'attribut prend la forme ***attribute='value'*** ou la forme *attribute*="*value*" où *attribute* et *value* sont respectivement le nom et la valeur de l'attribut.

  Chaque balise ouvrante peut contenir zéro ou plusieurs associations de valeurs à des attributs comme dans les exemples ci-dessous.

  <tag *attribute="value*"> ... </tag>

  <tag *attribute1="value1" attribute2="value2*"> ... </tag>

  Voici ci-dessous d'autres exemples de balises ouvrantes avec des attributs.

  <body background='yellow'>

  <xsd:element name="bibliography" type="Bibliography">

  <a href="#{$node/@idref}">

  Lorsque le contenu de l'élément est vide et que la balise ouvrante et la balise fermante sont contractées en une seule balise, celle-ci peut contenir des attributs comme la balise ouvrante.

  <hr style="color:red; height:15px; width:350px;" />

  <xsd:attribute name="key" type="xsd:NMTOKEN" use="required"/>

  Le nom de chaque attribut doit être un nom XML.

  La **valeur d'un attribut** peut être une chaîne quelconque de caractères délimitée par une paire d'apostrophes " ou une paire de guillemets " ".

  Elle peut contenir les caractères spéciaux '<', '>', '&', " et " " mais ceux-ci doivent nécessairement être introduits par les **entités prédéfinies**.

  Si la valeur de l'attribut est délimitée par des apostrophes ", les guillemets "" peuvent être introduits directement sans entité et inversement.

<xsl:value-of select="key('idchapter', @idref)/title"/>

<xsl:if test='@quote = "'"'>

Comme des espaces peuvent être présents dans la balise après le nom de l'élément et entre les attributs, l'indentation est libre pour écrire les attributs d'une balise ouvrante.

Aucun espace ne peut cependant séparer le caractère '=' du nom de l'attribut et de sa valeur.

Il est ainsi possible d'écrire l'exemple générique suivant.

<tag *attribute1*="*value1*"
***attribute2*="*value2*"**
...
***attributeN*="*valueN*">**
...
</tag>

L'ordre des attributs n'a pas d'importance. Les attributs d'un élément doivent avoir des **noms distincts**. Il est donc impossible d'avoir deux occurrences du même attribut dans une même balise ouvrante.

Le bon usage des attributs est pour les meta-données plutôt que les données elles-mêmes. Ces dernières doivent être placées de préférence dans le contenu des éléments.

Dans l'exemple suivant, la date proprement dite est placée dans le contenu alors que l'attribut format précise son format.

La norme ISO 8601 spécifie la représentation numérique de la date et de l'heure.

<date format="ISO-8601">2009-01-08</date>

Le nom complet d'un individu peut, par exemple, être réparti entre des éléments **firstname** et **surname** regroupés dans un élément **personname** comme dans l'exemple ci-dessous.

<personname id="I666">
<firstname>Gaston</firstname>
<surname>Lagaffe</surname>
</personname>

Les éléments **firstname** et **surname** peuvent être remplacés par des attributs de l'élément **personname** comme dans l'exemple ci-dessous.

Les deux solutions sont possibles mais la **première est préférable**.

<personname id="I666" firstname="Gaston" surname="Lagaffe"/>

- ***Attributs particuliers***

Il existe quatre attributs particuliers ***xml:lang, xml:space, xml:base et xml:id*** qui font partie de l'espace de noms XML.

Lors de l'utilisation de schémas, ces attributs peuvent être déclarés en important le schéma à l'adresse http://www.w3.org/2001/xml.xsd.

Contrairement à l'attribut xml:id, les trois autres attributs xml:lang, xml:space et xml:base s'appliquent au contenu de l'élément. Pour cette raison, la valeur de cet attribut est héritée par les enfants et, plus généralement, les descendants.

Ceci ne signifie pas qu'un élément dont le père a, par exemple, un attribut xml:lang a également un attribut xml:lang.

Cela veut dire qu'une application doit prendre en compte la valeur de l'attribut xml:lang pour le traitement l'élément mais aussi de ses descendants à l'exception, bien sûr, de ceux qui donnent une nouvelle valeur à cet attribut.

Autrement dit, la valeur de l'attribut xml:lang à prendre en compte pour le traitement d'un élément est celle donnée à cet attribut par l'ancêtre (y compris l'élément lui-même) le plus proche.

Pour illustrer le propos, le document suivant contient plusieurs occurrences de l'attribut xml:lang. La langue du texte est, à chaque fois, donnée par la valeur de l'attribut xml:lang le plus proche.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<book xml:lang="fr">
        <chapter>
        <title>Chapitre en Français</title>
        <p>Paragraphe en Français</p>
        <p xml:lang="en">Paragraph in English</p>
        </chapter>
```

```
        <chapter xml:lang="en">
        <title>Chapiter in English</title>
        <p xml:lang="fr">Paragraphe en Français</p>
        <p>Paragraph in English</p>
        </chapter>
</book>
```

Ce qui a été expliqué pour l'attribut xml:lang vaut également pour deux autres attributs xml:space et xml:base. C'est cependant un peu différent pour l'attribut xml:base car la valeur à prendre en compte doit être calculée à partir de toutes les valeurs des attributs xml:base des ancêtres.

- ***Attribut xml:lang***

L'attribut **xml:lang** est utilisé pour décrire la langue du contenu de l'élément.

Sa valeur est un code de langue sur deux ou trois lettres de la norme ISO 639 (comme par exemple en, fr, es, de, it, pt, …). Ce code peut être suivi d'un code de pays sur deux lettres de la norme ISO 3166 séparé du code de langue par un caractère tiret '-'.

L'attribut xml:lang est du type xsd:language qui est spécialement prévu pour cet attribut.

```
<p xml:lang="fr">Bonjour</p>
<p xml:lang="en-GB">Hello</p>
<p xml:lang="en-US">Hi</p>
```

Dans le document donné en exemple au début du chapitre, chaque élément book a un attribut lang. Ce n'est pas l'attribut xml:lang qui a été utilisé car celui-ci décrit la langue des données contenues dans l'élément alors que l'attribut lang décrit la langue du livre référencé.

- ***Attribut xml:space***

L'attribut **xml:space** permet d'indiquer à une application le traitement des caractères d'espacement. Les deux valeurs possibles de cet attribut sont **default** et **preserve**.

Un retour à la ligne est vu comme un simple espace.

Plusieurs espaces consécutifs sont aussi considérés comme un seul espace.

Si l'attribut xml:**space** a la valeur **preserve**, l'application doit respecter les caractères d'espacement.

Les retours à la ligne sont préservés et les espaces consécutifs ne sont pas confondus.

- ***Attribut xml:base***

  À chaque élément d'un document XML est associée une URI appelée *URI de base*.

  Celle-ci est utilisée pour résoudre les URL des entités externes, qui peuvent être, par exemple des fichiers XML ou des fichiers multimédia (images, sons, vidéo).

  L'attribut xml:base permet de préciser l'URL de base de l'élément. Cette URL peut être une URL **complète** ou une **adresse relative**.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"
?>
<book
xml:base="http://www.somewhere.org/Teaching/index.html">1
<chapter xml:base="XML/chapter.html">2
<section xml:base="XPath/section.html"/>3
<section xml:base="/Course/section.html"/>4
<section xml:base="http://www.elsewhere.org/section.html"/>5
</chapter>
</book>
1 http://www.somewhere.org/Teaching/index.html
2 http://www.somewhere.org/Teaching/XML/chapter.html
3 http://www.somewhere.org/Teaching/XML/XPath/section.html
4 http://www.somewhere.org/Course/section.html
5 http://www.elsewhere.org/section.html
```

- ***Attribut xml:id***

  L'attribut **xml:id** permet d'associer un identificateur à tout élément indépendamment de toute DTD ou de tout schéma.

4.3 *Élément racine*

Tout le corps du document doit être compris dans le contenu d'un unique élément appelé ***élément racine***.

Le nom de cet élément racine est donné par la déclaration de type de document si celle-ci est présente.

L'élément **bibliography** est l'élément racine de l'exemple donné au début du chapitre.

```
  ...              ]  Commentaires et instructions de traitement  ]
<root-element>     ]  Balise ouvrante                             |   Corps
  ...              ]  Éléments, commentaires et                   |    du
  ...              ]  instructions de traitement                  |  document
</root-element>    ]  Balise fermante                             |
  ...              ]  Commentaires et instructions de traitement  ]
```

4.4 *Commentaires*

Les commentaires sont délimités par les chaînes de caractères '<!--' et '-->' comme en HTML.

Ils ne peuvent pas contenir la chaîne '--' formée de deux tirets '-' et ils ne peuvent donc pas être imbriqués.

Ils peuvent être présents dans le prologue et en particulier dans la DTD.

Ils peuvent aussi être placés dans le contenu de n'importe quel élément et après l'élément racine.

Un exemple de document XML avec des commentaires partout où ils peuvent apparaître est donné ci-dessous.

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
    <!-- Commentaire dans le prologue avant la DTD -->
    <!DOCTYPE simple [
    <!-- Commentaire dans la DTD -->
    <!ELEMENT simple (#PCDATA) >
    ]>
    <!-- Commentaire entre le prologue et le corps du document -->
    <simple>
```

<!-- Commentaire au début du contenu de l'élément simple -->
Un exemple simplissime
<!-- Commentaire à la fin du contenu de l'élément simple -->
</simple>
<!-- Commentaire après le corps du document -->

Les caractères spéciaux '<', '>' et '&' peuvent apparaître dans les commentaires. Il est en particulier possible de mettre en commentaire des éléments avec leurs balises :

<!-- <tag type="comment">Élément mis en commentaire</tag> -->

5. Exemples

5.1 Exemple 1

L'exemple suivant contient uniquement un prologue avec la déclaration XML et un élément de contenu vide.

Ce document n'a pas de déclaration de DTD.

<?xml version="1.0"?>

<tag/>

5.2 Exemple 2

Cet exemple contient une déclaration de DTD qui permet de valider le document. Cette DTD définit l'élément simple et déclare que son contenu doit être textuel.

<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

<!DOCTYPE simple [

<!ELEMENT simple (#PCDATA) >]>

<simple>Un exemple simplissime</simple>

6. ***Xinclude***

Il est possible de répartir un gros document en plusieurs fichiers afin d'en rendre la gestion plus aisée.

Il existe essentiellement deux méthodes pour atteindre cet objectif.

Le point commun de ces méthodes est de scinder le document en différents fichiers qui sont ***inclus*** par un fichier principal.

XInclude définit un élément **xi:include** dans un espace de noms associé à l'URL.

Cet élément a un attribut **href** qui contient le nom du fichier à inclure et un attribut **parse** qui précise le type des données.

Cet attribut peut prendre les valeurs **xml** ou **text**.

Le fichier source principal de cet ouvrage inclut, par exemple, les fichiers contenant les différents chapitres grâce à des éléments **include** comme ci-dessous.

```
<book version="5.0"
xmlns="http://docbook.org/ns/docbook"
xmlns:xi="http://www.w3.org/2001/XInclude">
...
<!-- Inclusion des différents chapitres -->
<xi:include href="introduction.xml" parse="xml"/>
<xi:include href="Syntax/chapter.xml" parse="xml"/>
...
</book>
```

Le fragment de document contenu dans un fichier inclus doit être bien formé.

Il doit en outre être entièrement contenu dans un seul élément qui est l'élément racine du fragment.

Le chemin d'accès au fichier est récupéré dans l'attribut **href** de l'élément xi:include.

La mise à jour des attributs xml:base garde une trace des inclusions et permet aux liens relatifs de rester valides.

```
<book version="5.0"
xmlns="http://docbook.org/ns/docbook"
xmlns:xi="http://www.w3.org/2001/XInclude">

...

<!-- Inclusion des différents chapitres -->
<chapter xml:id="chap.introduction"
xml:base="introduction.xml">

...

</chapter>
<chapter xml:id="chap.syntax" xml:base="Syntax/chapter.xml">
...

</chapter>
...
</book>
```

7. DTD

Le rôle d'une **DTD** (*Document Type Definition*) est de définir précisément la structure d'un document.

Il s'agit d'un certain nombre de contraintes que doit respecter un document pour être ***valide***.

Ces contraintes spécifient quelles sont les éléments qui peuvent apparaître dans le contenu d'un élément, l'ordre éventuel de ces éléments et la présence de texte brut.

Elles définissent aussi, pour chaque élément, les attributs autorisés et les attributs obligatoires.

Les DTD ont l'avantage d'être relativement simples à utiliser mais elles sont parfois aussi un peu limitées.

Les schémas XML permettent de décrire de façon plus précise encore la structure d'un document.

Exemple :

On reprend la petite bibliographie du fichier bibliography.xml.

La troisième ligne de ce fichier est la déclaration de la DTD qui référence un fichier externe **bibliography.dtd.**

Le nom **bibliography** de l'élément racine du document apparaît dans cette déclaration juste après le mot clé DOCTYPE.

```
<!DOCTYPE bibliography SYSTEM "bibliography.dtd" >
<!ELEMENT bibliography (book)+ > 1
<!ELEMENT book (title, author, year, publisher, isbn, url?) > 2
<!ATTLIST book key NMTOKEN #REQUIRED > 3
<!ATTLIST book lang (fr | en) #REQUIRED > 4
<!ELEMENT title (#PCDATA) > 5
<!ELEMENT author (#PCDATA) >
<!ELEMENT year (#PCDATA) >
<!ELEMENT publisher (#PCDATA) >
<!ELEMENT isbn (#PCDATA) >
<!ELEMENT url (#PCDATA) >
```

1 Déclaration de l'élément **bibliography** devant contenir une suite non vide d'éléments **book**.

2 Déclaration de l'élément **book** devant contenir les éléments **title**, **author**, …, **isbn** et **url**.

3 et 4 Déclarations des attributs obligatoires **key** et **lang** de l'élément **book**.

5 Déclaration de l'élément **title** devant contenir uniquement du texte.

7.1 Déclaration de la DTD

La déclaration de la **DTD** du document doit être placée dans le **prologue**.

La DTD peut être interne, externe ou mixte.

Elle est ***interne*** si elle est directement incluse dans le document.

Elle est ***externe*** si le document contient seulement une référence vers un autre document contenant la DTD.

Elle est finalement mixte si elle est constituée d'une partie interne et d'une partie externe.

Une DTD est généralement prévue pour être utilisée pour de multiples documents.

La déclaration de la DTD est introduite par le mot clé **DOCTYPE** et a la forme générale suivante où ***root-element*** est le nom de l'élément racine du document.

<!DOCTYPE *root-element* ... >

- ***DTD interne***

Lorsque la DTD est incluse dans le document, sa déclaration prend la forme suivante où son contenu est encadré par des crochets '[' et ']'.

<!DOCTYPE *root-element* [*declarations*] >

Les déclarations ***declarations*** constituent la définition du type du document.

Dans l'exemple suivant de DTD, le nom de l'élément racine est **simple**.

La DTD déclare en outre que cet élément ne peut contenir que du texte (*Parsed Characters DATA*) et pas d'autre élément.

<!DOCTYPE simple [

<!ELEMENT simple (#PCDATA) >

]>

- ***DTD externe***

Lorsque la DTD est externe, celle-ci est contenue dans un autre fichier dont l'extension est généralement **.dtd**.

Le document XML se contente alors de donner l'adresse de sa DTD pour que les logiciels puissent y accéder.

L'adresse de la DTD peut être donnée explicitement par une URL ou par un FPI (*Formal Public Indentifier*).

Les FPI sont des noms symboliques donnés aux documents.

Ils sont utilisés avec des catalogues qui établissent les correspondances entre ces noms symboliques et les adresses réelles des documents.

Lorsqu'un logiciel rencontre un FPI, il parcourt le catalogue pour le *résoudre*, c'est-à-dire déterminer l'adresse réelle du document.

Les catalogues peuvent contenir des adresses locales et/ou des URL.

Adressée par FPI

Les FPI (*Formal Public Identifier*) sont des identifiants de document hérités.

Ils sont plutôt remplacés en XML par les URI qui jouent le même rôle.

Ils sont constitués de quatre parties séparées par des chaînes '//' et ils obéissent donc à la syntaxe suivante.

type//*owner*//*desc*//*lang*

Le premier caractère ***type*** du FPI est soit le caractère '+' si le propriétaire est enregistré selon la norme ISO 9070 soit le caractère '-' sinon.

Le FPI continue avec le propriétaire ***owner*** et la description ***desc*** du document.

Le FPI se termine par un code de langue *lang* de la norme ISO 639.

Des exemples de FPI sont donnés ci-dessous.

-//W3C//DTD XHTML 1.0 Strict//EN

-//OASIS//DTD Entity Resolution XML Catalog V1.0//EN

ISO/IEC 10179:1996//DTD DSSSL Architecture//EN

Un FPI peut être converti en URI en utilisant l'espace de noms publicid des URN et en remplaçant chaque chaîne '//' par le caractère ':' et chaque espace par le caractère '+' comme dans l'exemple ci-dessous.

urn:publicid:-:W3C:DTD+XHTML+1.0+Strict:EN

- ***Adressée par URL***

  La référence à une URL est introduite par le mot clé SYSTEM suivi de l'URL délimitée par des apostrophes '" ou des guillemets "".

  <!DOCTYPE *root-element* SYSTEM "*url*" >

  L'URL *url* peut être soit une URL complète commençant par http:// ou ftp:// soit plus simplement le nom d'un fichier local comme dans les exemples suivants.

  <!DOCTYPE bibliography SYSTEM

  "http://www.liafa.jussieu.fr/~carton/Enseignement/bibliography.dtd">

  <!DOCTYPE bibliography SYSTEM "bibliography.dtd">

- ***DTD mixte***

  Il est possible d'avoir une DTD externe adressée par URL ou FPI et des déclarations internes.

  La déclaration prend alors une des deux formes suivantes : On retrouve un mélange de la syntaxe des DTD externes avec les mots clés SYSTEM et PUBLIC et de la syntaxe des DTD internes avec des déclarations encadrées par les caractères '[' et ']'.

  <!DOCTYPE *root-element* SYSTEM "*url*" [*declarations*] >

  <!DOCTYPE *root-element* PUBLIC "*fpi*" "*url*" [*declarations*] >

  La définition interne a alors priorité sur la définition externe.

  Ce mécanisme permet d'utiliser une DTD externe tout en adaptant certaines définitions au document.

7.2 Contenu de la DTD

Une DTD est essentiellement constituée de déclarations d'éléments et d'attributs. Elle peut aussi contenir des déclarations d'entités qui sont des ***macros*** semblables aux **#define** du langage C.

- ***Commentaires***

Une DTD peut contenir des commentaires.

Ceux-ci sont placés au même niveau que les déclarations d'éléments, d'attributs et d'entités.

<!-- DTD pour les bibliographies -->

<!ELEMENT bibliography (book)+ >

<!ELEMENT book (title, author, year, publisher, isbn, url?) >

...

- ***Déclaration d'entité***

Une ***entité*** est un nom donné à un fragment de document.

Ce fragment peut être inséré dans le document en utilisant simplement le nom de l'entité.

Si l'entité a pour nom ***entity***, le fragment est inséré par **&*entity*;** où le nom de l'entité est encadré des caractères '&' et ';'.

L'entité peut être utilisée dans le contenu des éléments et dans les valeurs des attributs comme dans l'exemple ci-dessous.

<tag meta="attribute: &*entity*;">Content: &*entity*;</tag>

Quelques entités sont prédéfinies afin d'insérer les caractères spéciaux.

D'autres entités peuvent être définies à l'intérieur de la DTD du document.

- ***Entités prédéfinies***

Il existe des entités prédéfinies permettant d'inclure les caractères spéciaux '<', '>', '&', ''' et '"' dans les contenus d'éléments et dans les valeurs d'attributs.

Ces entités sont les suivantes.

Entité	Caractère
<	<
>	>
&	&
'	'
"	"

Tableau 3.1. Entités prédéfinies

- ***Entité interne***

Une entité est dite ***interne*** lorsque le fragment est inclus directement dans le document.

La déclaration d'une telle entité prend la forme suivante où l'identifiant ***entity*** est le nom l'entité et *fragment* est la valeur de l'entité.

Cette valeur doit être un fragment XML bien formé. Elle peut contenir des caractères et des éléments.

<!ENTITY *entity* "*fragment*" >

Si la DTD contient par exemple la déclaration d'entité suivante.

<!ENTITY aka "also known as" >
<!ENTITY euro "&#20AC;" >

Il est possible d'inclure le texte *also known* as en écrivant seulement *&aka*;.

- ***Entité externe***

Une entité peut désigner une fraction de document contenu dans un autre fichier. Ce mécanisme permet de répartir un même document sur plusieurs fichiers.

La déclaration utilise alors le mot clé **SYSTEM** suivi d'une URL qui peut simplement être le nom d'un fichier local. Le fichier principal inclut les différentes parties en définissant une entité externe pour chacune de ces parties.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE book [
<!-- Entités externes -->
<!ENTITY chapter1 SYSTEM "chapter1.xml" >
<!ENTITY chapter2 SYSTEM "chapter2.xml" >]>
<book>
&chapter1;
&chapter2;
</book>
```

- ***Entité paramètre***

Les *entités paramètres* sont des entités qui peuvent uniquement être utilisées à l'intérieur de la DTD.

La déclaration d'une entité paramètre prend la forme suivante.

<!ENTITY % *entity* "*fragment*" >

L'entité ***entity*** ainsi déclarée peut être utilisée en écrivant %*entity*; où le nom de l'entité est encadré des caractères '%' et ';'.

Une entité paramètre peut uniquement être utilisée dans la partie externe de la DTD.

L'exemple suivant définit deux entités paramètre idatt et langatt permettant de déclarer des attributs id et xml:lang facilement.

```
<!ENTITY % idatt "id ID #REQUIRED" >
<!ENTITY % langatt "xml:lang NMTOKEN 'fr'" >
<!ATTLIST chapter %idatt; %langatt; >
<!ATTLIST section %langatt; >
```

- ***Déclaration d'élément***

La déclaration d'un élément est nécessaire pour qu'il puisse apparaître dans un document.

Cette déclaration précise le nom et le type de l'élément.

Le nom de l'élément doit être un nom XML et le type détermine les contenus valides de l'élément.

On distingue :

Les *contenus purs* uniquement constitués d'autres éléments.

Les *contenus textuels* uniquement constitués de texte.

Les *contenus mixtes* qui mélangent éléments et texte.

De manière générale, la déclaration d'un élément prend la forme suivante où *element* et *type* sont respectivement le nom et le type de l'élément.

<!ELEMENT *element type* >

- ***Contenu pur d'éléments***

Lorsque le contenu d'un élément est pur, celui-ci ne peut pas contenir de texte mais seulement d'autres éléments.

Ces éléments fils peuvent, à leur tour, contenir d'autres éléments et/ou du texte.

Leur contenu est spécifié par leur propre déclaration dans la DTD.

La déclaration de l'élément détermine quels éléments il peut contenir directement et dans quel ordre. Une déclaration d'élément a la forme suivante.

<!ELEMENT *element regexp* >

Le nom de l'élément est donné par l'identifiant *element* et l'expression rationnelle *regexp* décrit les suites autorisées d'éléments dans le contenu de l'élément.

Cette expression rationnelle est construite à partir des noms d'éléments en utilisant les opérateurs ',', '|', '?', '*' et '+' ainsi que les parenthèses '(' et ')' pour former des groupes.

Les opérateurs ',' et '|' sont binaires alors que les opérateurs '?', '*' et '+' sont unaires et postfixés.

Ils se placent juste après leur opérande.

Opérateur	Signification
,	Mise en séquence
|	Choix
?	0 ou 1 occurrence
*	Itération (nombre quelconque d'occurrences)
+	Itération stricte (nombre non nul d'occurrences)

Opérateurs des DTD

Cette définition est illustrée par les exemples suivants.

<!ELEMENT elem (elem1, elem2, elem3) >

L'élément elem doit contenir un élément elem1, un élément elem2 puis un élément elem3 dans cet ordre.

<!ELEMENT elem (elem1 | elem2 | elem3) >

L'élément elem doit contenir un seul des éléments elem1, elem2 ou elem3.

<!ELEMENT elem (elem1, elem2?, elem3) >

L'élément elem doit contenir un élément elem1, un ou zéro élément elem2 puis un élément elem3 dans cet ordre.

<!ELEMENT elem (elem1, elem2*, elem3) >

L'élément elem doit contenir un élément elem1, une suite éventuellement vide d'éléments elem2 et un élément elem3 dans cet ordre.

<!ELEMENT elem (elem1, (elem2 | elem4), elem3) >

L'élément elem doit contenir un élément elem1, un élément elem2 ou un élément elem4 puis un élément elem3 dans cet ordre.

<!ELEMENT elem (elem1, elem2, elem3)* >

L'élément elem doit contenir une suite d'éléments elem1, elem2, elem3, elem1, elem2, … jusqu'à un élément elem3.

<!ELEMENT elem (elem1 | elem2 | elem3)* >

L'élément elem doit contenir une suite quelconque d'éléments elem1, elem2 ou elem3.

<!ELEMENT elem (elem1 | elem2 | elem3)+ >

L'élément elem doit contenir une suite non vide d'éléments elem1, elem2 ou elem3.

- ***Contenu textuel***

La déclaration de la forme suivante indique qu'un élément peut uniquement contenir du texte.

Ce texte est formé de caractères, d'entités qui seront remplacées au moment du traitement et de sections littérales CDATA.

<!ELEMENT *element* (#PCDATA) >
Dans l'exemple suivant, l'élément text est de type #PCDATA.
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE texts [<!ELEMENT texts (text)* >
<!ELEMENT text (#PCDATA) >
]>
<texts>
<text>Du texte simple</text>
<text>Une <![CDATA[Section CDATA avec < et >]]></text>
<text>Des entités < et ></text>
</texts>

- ***Contenu mixte***

La déclaration de la forme suivante indique qu'un élément peut uniquement contenir du texte et les éléments element1, …, elementN.

Dans une telle déclaration, le mot clé #PCDATA doit apparaître en premier avant tous les noms des éléments.

<!ELEMENT *element* (#PCDATA | *element1* | ... | *elementN*)* >

Dans l'exemple suivant, l'élément **book** possède un contenu mixte.

<?xml version="1.0" encoding="iso-8859-1" ?>

```
<!DOCTYPE book [
<!ELEMENT book (#PCDATA | em | cite)* >
<!ELEMENT em (#PCDATA) >
<!ELEMENT cite (#PCDATA) >]>
<book>
Du <em>texte</em>, une <cite>citation</cite> et encore du <em>texte</em>.
</book>
```

- ***Contenu vide***

La déclaration suivante indique que le contenu de l'élément element est nécessairement vide.

Cet élément peut uniquement avoir des attributs.

Les éléments vides sont souvent utilisés pour des liens entre éléments.

<!ELEMENT *element* EMPTY >

- ***Contenu libre***

La déclaration suivante n'impose aucune contrainte sur le contenu de l'élément *element*.

Ce type de déclarations permet de déclarer des éléments dans une DTD en cours de mise au point afin de procéder à des essais de validation.

<!ELEMENT *element* ANY >

- ***Déclaration d'attribut***

La déclaration d'attribut prend la forme générale suivante où *attribut* est le nom de l'attribut et *element* le nom de l'élément auquel il appartient.

Cette déclaration comprend également le type *type* et la valeur par défaut *default* de l'attribut.

Le nom de l'attribut doit être un nom XML.

<!ATTLIST *element attribut type default* >

Il est possible de déclarer simultanément plusieurs attributs pour un même élément.

<!ATTLIST ***element attribut1 type1 default1***
attribut2 type2 default2
...
attributN typeN defaultN **>**

Type des attributs

Le type d'un attribut détermine quelles sont ses valeurs possibles.

Les DTD proposent uniquement un choix fini de types pour les attributs.

Les types les plus utilisés sont : CDATA, ID et IDREF, …

CDATA : Ce type est le plus général. Il n'impose aucune contrainte à la valeur de l'attribut. Celle-ci peut être une chaîne quelconque de caractères. (*value1* | *value2* | ... | *valueN*)

ID : La valeur de l'attribut est un nom XML. Un élément peut avoir un seul attribut de ce type.

IDREF : La valeur de l'attribut est une référence à un élément identifié par la valeur de son attribut de type ID.

IDREFS : La valeur de l'attribut est une liste de références séparées par des espaces.

NOTATION : La valeur de l'attribut est une notation

▪ ***Outils de validation***

Il existe plusieurs outils permettant de valider un document XML par rapport à une DTD. Il existe d'abord des sites WEB.

- Page de validation du W3C [http://validator.w3.org/]
- Page de validation du Scholarly Technology Group de l'université de Brown [http://www.stg.brown.edu/service/xmlvalid/]

L'inconvénient majeur de ces sites WEB est la difficulté de les intégrer à une chaîne de traitement automatique puisqu'ils requièrent l'intervention de l'utilisateur.

Dans ce cas, il est préférable d'utiliser un logiciel. Avec l'option --valid, il réalise la validation du document passé en paramètre.

Avec cette option, la DTD soit être précisée par une déclaration de DTD dans le prologue du document.

Sinon, il faut donner explicitement donner la DTD après l'option --dtdvalid.

8. Espaces de noms

Identification d'un espace de noms

Un *espace de noms* est identifié par un URI appelé *URI de l'espace de noms*.

Cet URI garantit seulement que l'espace de noms soit identifié de manière unique.

Déclaration d'un espace de noms

Un espace de noms déclaré par un pseudo attribut de forme xmlns:*prefix* dont la valeur est une URL qui identifie l'espace de noms.

Le préfixe *prefix* est un nom XML ne contenant pas le caractère ':'.

Il est ensuite utilisé pour *qualifier* les noms d'éléments.

Un *nom qualifié* d'élément prend la forme *prefix*:*local* où *prefix* est un préfixe associé à un espace de noms et *local* est le *nom local* de l'élément.

Quelques espaces de noms classiques

XML : http://www.w3.org/XML/1998/namespace

XInclude : http://www.w3.org/2001/XInclude

XLink : http://www.w3.org/1999/xlink

MathML : http://www.w3.org/1998/Math/MathML

XHTML : http://www.w3.org/1999/xhtml

SVG : http://www.w3.org/2000/svg

Schémas : http://www.w3.org/2001/XMLSchema

Instances de schémas ; http://www.w3.org/2001/XMLSchema-instance

XSLT : http://www.w3.org/1999/XSL/Transform

XSL-FO : http://www.w3.org/1999/XSL/Format

DocBook : http://docbook.org/ns/docbook

Schematron : http://purl.oclc.org/dsdl/schematron

9. *Schémas XML*

Les *schémas XML* permettent comme les DTD de définir des modèles de documents. Il est ensuite possible de vérifier qu'un document donné respecte un schéma.

▪ *Structure globale d'un schéma*

Un schéma XML se compose essentiellement de déclarations d'éléments et d'attributs et de définitions de types.

Chaque élément est déclaré avec un type qui peut être, soit un des types prédéfinis, soit un nouveau type défini dans le schéma.

Le type spécifie quels sont les contenus valides de l'élément ainsi que ses attributs.

L'espace de noms des schémas XML est identifié par l'URL **http://www.w3.org/2001/XMLSchema.**

Il est généralement associé à xsd ou à xs.

La structure globale d'un schéma est donc la suivante.

<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<!-- Déclarations d'éléments, d'attributs et définitions de types -->
...
</xsd:schema>

Les éléments, attributs et les types peuvent être *globaux* ou *locaux*.

Ils sont globaux lorsque leur déclaration ou leur définition est enfant direct de l'élément xsd:schema. Sinon, ils sont locaux.

▪ ***Types prédéfinis***

Les schémas possèdent de nombreux types prédéfinis.

Certains, comme xsd:int et xsd:float, proviennent des langages de programmation, certains, comme xsd:date et xsd:time, sont inspirés de normes ISO (ISO 8601 dans ce cas) et d'autres encore, comme xsd:ID, sont hérités des DTD. Ces types autorisent l'écriture de schémas concis et très précis.

- ***Types numériques***

xsd:int ou xsd:double correspondent à un codage précis et donc à une précision fixée alors que d'autres types comme xsd:integer ou xsd:decimal autorisent une précision arbitraire.

xsd:boolean Valeur booléenne avec true ou 1 pour *vrai* et false ou 0 pour *faux*

xsd:byte Nombre entier signé sur 8 bits

xsd:unsignedByte Nombre entier non signé sur 8 bits

xsd:short Nombre entier signé sur 16 bits

xsd:unsignedShort Nombre entier non signé sur 16 bits

xsd:int Nombre entier signé sur 32 bits

xsd:unsignedInt Nombre entier non signé sur 32 bits

xsd:long Nombre entier signé sur 64 bits. Ce type dérive du type xsd:integer. xsd:unsignedLong Nombre entier non signé sur 64 bits

xsd:integer Nombre sans limite de précision. Ce type n'est pas primitif et dérive du type xsd:decimal.

xsd:positiveInteger Nombre entier strictement positif sans limite de précision

xsd:negativeInteger Nombre entier strictement négatif sans limite de précision

xsd:nonPositiveInteger Nombre entier négatif ou nul sans limite de précision

xsd:nonNegativeInteger Nombre entier positif ou nul sans limite de précision

xsd:float Nombre flottant sur 32 bits conforme à la norme IEEE 754 [ ]

xsd:double Nombre flottant sur 64 bits conforme à la norme IEEE 754 [ ]

xsd:decimal Nombre décimal sans limite de précision

- ***Types pour les chaînes et les noms***

Les schémas possèdent bien sûr un type pour les chaînes de caractères ainsi que quelques types pour les noms qualifiés et non qualifiés.

xsd:string Chaîne de caractères composée de caractères Unicode

xsd:normalizedString Chaîne de caractères normalisée.

xsd:token Chaîne de caractères normalisée (comme ci-dessus) et ne contenant pas en outre des espaces en début ou en fin ou des espaces consécutifs

xsd:Name Nom XML.

…

- ***Types pour les dates et les heures***

xsd:time Heure au format hh:mm:ss[.sss][TZ], par exemple 14:07:23. La partie fractionnaire .sss des secondes est optionnelle.

Tous les autres champs sont obligatoires.

Les nombres d'heures, minutes et de secondes doivent être écrits avec deux chiffres en complétant avec 0.

L'heure peut être suivie d'un décalage horaire TZ qui est soit Z pour le temps universel soit un décalage commençant par + ou - comme -07:00.

xsd:date Date au format YYYY-MM-DD, par exemple 2008-01-16. Tous les champs sont obligatoires.

xsd:dateTime Date et heure au format YYYY-MM-DDThh:mm:ss, par exemple 2008-01-16T14:07:23. Tous les champs sont obligatoires.

xsd:duration Durée au format PnYnMnDTnHnMnS

- ***Types hérités des DTD***

xsd:ID nom XML identifiant un élément

xsd:IDREF référence à un élément par son identifiant

xsd: IDREFS liste de références à des éléments par leurs identifiants

xsd:ENTITY entité externe non XML

xsd:ENTITIES liste d'entités externes non XML separés par des espaces

xsd:NOTATION notation

- ***Types simples***

Les types simples définissent uniquement des contenus textuels. Ils peuvent être utilisé pour les éléments ou les attributs. Ils sont introduits par l'élément xsd:simpleType.

Un type simple est souvent obtenu par restriction d'un autre type défini.

Il peut aussi être construit par union d'autres types simples ou par l'opérateur de listes.

La déclaration d'un type simple a la forme suivante.

```
<xsd:simpleType ...>
...
</xsd:simpleType>
```

L'élément xsd:simpleType peut avoir un attribut name si la déclaration est globale.

```
<xsd:simpleType name="Byte">
<xsd:restriction base="xsd:nonNegativeInteger">
<xsd:maxInclusive value="255"/>
</xsd:restriction>
</xsd:simpleType>
```

- ***Types complexes***

Les types complexes définissent des contenus purs (constitués uniquement d'éléments), des contenus textuels ou des contenus mixtes. Tous ces contenus peuvent comprendre des attributs. Les types complexes peuvent seulement être utilisés pour les éléments. Ils sont introduits par l'élément xsd:complexType.

La construction explicite d'un type se fait en utilisant les opérateurs de séquence **xsd:sequence**, de choix **xsd:choice** ou d'ensemble **xsd:all**.

La construction du type se fait directement dans le contenu de l'élément **xsd:complexType** et prend donc la forme suivante.

```
<!-- Type explicite -->
<xsd:complexType ...>
<!-- Construction du type avec xsd:sequence, xsd:choice ou xsd:all -->
...
</xsd:complexType>
```

Chapitre 3

PHP

INTRODUCTION

- PHP est un langage interprété orienté Web. Syntaxiquement, c'est un mélange de C et de Perl. Les scripts PHP sont lus et interprétés par le moteur PHP.
- PHP comporte plus de 500 fonctions. Il est fournit avec des librairies offrant des fonctionnalités diverses :
  - accès aux bases de données, (utilisation la plus courante de PHP)
  - fonctions d'images,
  - sockets,
  - protocoles Internet divers...

Notions essentielles

- Il faut bien distinguer le client et le serveur (imaginez tout bêtement la scène dans un café). Votre navigateur est le client, C'est lui qui demande la page web que vous avez entrée. Le serveur est l'ordinateur sur l'Internet qui héberge cette page web. PHP s'exécute donc côté serveur.
- En voici quelques conséquences :
  - Tout ce qui a trait à la présentation de la page (couleur du texte, etc..) est à faire en HTML, exécutés côté client. PHP n'a rien à voir avec le design de votre page
  - L'intérêt de PHP est de générer du HTML ou du Javascript dynamiquement. Le travail effectué avec PHP sur votre page est totalement invisible pour le visiteur.

PRINCIPE de FONCTIONNEMENT

- Le serveur web est un "ordinateur" présent sur l'Internet et qui héberge la page que vous demandez. Sur ce serveur on trouve Apache, logiciel apte à traiter les requêtes HTTP que vous envoyez lorsque vous demandez une page web. Apache va donc chercher le fichier demandé dans son arborescence et renvoie à votre navigateur la page HTML
- Ce code HTML est alors envoyé à travers le réseau au navigateur client. De plus, aucune ligne de code PHP n'apparaît côté client dans la mesure où tout le code a été interprété.

Le Langage PHP

- Le code PHP est toujours encadré par des balises le signalant. Les balises possibles sont :
- <?php ?>
- <? ?>
- <% %>
- <script language="php"> </script>
- Les plus couramment utilisés sont <? ?> , que vous trouverez dans beaucoup de scripts, même si elles ne sont pas les plus correctes. L'idéal pour éviter des problèmes futurs est d'utiliser les plus correctes : <?php ?> . Celles en <% %> sont à fuir le plus souvent possible, sauf en cas de nécessité de compatibilité avec un éditeur d'asp.

Le Langage PHP

- Le séparateur d'instructions est le ;.
  Il est obligatoire, sauf si l'instruction est suivie de la balise ?>
- La fonction **echo** affiche un (ou plus) argument. Si l'argument est une chaîne entre simple quotte ' il est affiché tel quel. echo 'Bonjour , Anas';

Le Langage PHP

Code PHP

```
<?php
    echo 'Bonjour Anas' ;
?>
<?php
    echo ' <font face="arial" size="2"
    color="red">Bonjour Anas </font> ';
?>
```

- PHP en rouge
- HTML en jaune

Affichage d'une image et du texte.

```
<?php
echo '<div align="center"><font face="arial"
size="2" color="blue">
Bonjour le monde !</font><br /> ';
echo '<img src="images/globe.gif" border="0"
/></div> ';
?>
```

Le Langage PHP

Affichage date et heure courante

```
<?php
  $date = date("d-m-Y");
  $heure = date("H:i");
  Print("Nous sommes le $date et il est
  $heure");
?>
```

Le Langage PHP

- Avec les double quottes " les variables contenues dans cette chaîne sont interprétées. $nom= "Anas ";
  - echo "Hello, $nom"; // Hello, Anas
  - echo 'Hello, $nom'; // Hello, $nom
- Remarque
  - echo ' j'utilise php '; // affiche une erreur
  - echo ' j\'utilise php '; // avant ‘ mettre un antislash

LES VARIABLES

- **Visibilité et affectation**
  - PHP n’est pas un langage fortement structuré, il ne contient donc pas de partie déclarative clairement définie. Pour définir une variable, il suffit de l’initialiser.
  - Les variables sont précédées du signe $, quelque soit leur type. Ainsi pour déclarer une variable **var** : $var=1;
  - La variable **$var** est alors définie et vaut 1. Elle devient immédiatement accessible et ce jusqu’à la fin du script.

Type de variables

- lors de l'affectation. Il existe six types de données :
  - Entier *(int, integer)*
    - *$nbr=450 ;*
  - Décimal *(real, float, double)*
    - *$montant=2500.25*
  - Chaîne de caractères *(string)*
    - *$article="PC Portable"*
  - Tableau *(array)*
  - Objet *(object)*
  - Booléen *(boolean, uniquement PHP4)*

LES VARIABLES

- Les opérateurs de conversion sont :
  - (string) conversion en chaîne de caractères
    - $nbr=254 $chaine_nbr=(string)$nbr /$chaine_nbr contient la chaine 254
  - (int) conversion en entier, synonyme de (integer)
    - $montant=(int)$chaine_nbr
  - (real) conversion en double, synonyme de (double) et (float)
    - $montant_dec=(real)$chaine_nbr
  - (array) conversion en tableau
  - (object) conversion en objet
  - (bool) conversion en booléen

Règles des conversions implicites :

- Si la chaîne de caractères contient un ***point***, un **e** ou un ***E*** ainsi que des ***caractères numériques***, elle est convertie en **décimal**,
- Si la chaîne de caractères ne contient que des ***caractères numériques***, elle est convertie en **entier**,
- Si la chaîne de caractères contient ***plusieurs mots***, seul le **premier est pris en compte** et est converti selon les règles ci-dessus.
  - $var1 = 1; // $var1 est de type "integer" et vaut 1.
  - $var2 = 12.0; // $var2 est de type "double" et vaut 12.
  - $var3 = "PHP"; // $var3 est de type "string" et vaut "PHP".
  - $var4 = false; // $var4 est de type "boolean" et vaut false.
  - $var5 = "5a"; // $var5 est de type "string" et vaut "5a".

Tests sur les variables

- La fonction **gettype** permet de connaître le type de la variable. Elle renvoie une chaîne : "string" ou "integer" ou "double" ou "array" ou "object".
  - ***Remarque** : Si la variable n'est pas définie, elle renvoie "string".*
  - $a= 12;
  - echo gettype($a) ; // => "integer"
  - $a= $a / 10;
  - echo gettype($a) ; // => "double"
  - unset($a);
  - echo gettype($a) ; // => "string"

- On peut également utiliser **strval**, **intval**, **doubleval** qui renvoient la variable convertie en chaîne / entier / réel.
  - $chainePI= "3.1415";
  - $intPI= intval($chainePI);
  - $PI= doubleval($chainePI);
  - echo " $chainePI / $intPI / $PI"; // => 3.1415 / 3 / 3.1415

Tests sur les variables

- On peut également tester un type particulier a l'aide des fonctions **is_array**, **is_string**, **is_int**, **is_float**, **is_object** .
  - $a= 123;
  - echo is_int($a); // => (vrai)
  - echo is_double($a) // => (faux)
  - echo is_string($a) // => (faux)
  - $a += 0.5;
  - echo is_float($a) // => (vrai)
- *Remarque : Les fonctions **is_double** et **is_real** sont équivalentes à **is_float**. Les fonctions **is_long** et **is_integer** sont équivalentes à **is_int**.*

Tests sur les variables

- La fonction **isset** permet de tester si une variable est définie. La fonction **unset** permet de supprimer la variable, et de désallouer la mémoire utilisée.
  - echo isset($a); // => 0 (faux)
  - $a= " ";
  - unset($a); // => 1 (vrai) la libération de la mémoire
  - echo isset($a); // => 0 (faux)

LES CONSTANTES

- PHP permet de définir des constantes a l'aide de la fonction **define**.
  - define("Couleur", "rouge");
- Deux constantes sont prédéfinies par PHP :
  - **__FILE__** contient le nom du fichier,
  - et **__LINE__** le numéro de la ligne courante.
  - define("NEXTPAGE", "script2.PHP");
  - echo "Page courante : ", __FILE__ , "Page suivante : ", NEXTPAGE;
- pas de $ pour des constantes.

Références

- PHP4 permet d'exploiter les références aux variables, à l'instar du langage C. Une référence à une variable est un accès à la zone mémoire qui contient la valeur de cette variable. Cette référence est désignée par le caractère & placé devant le nom de la variable.
  - $a = 1 ; // $a a pour valeur 1.
  - $b = &$a ;
  - $b et $a pointent sur la même zone mémoire.
  - Ce sont donc deux noms pour la même variable.
  - echo " $a, $b " ; // Affiche 1, 1
  - $a = 2 ;
  - echo " $a, $b " ; // Affiche 2, 2

LES OPERATEURS (1)

- PHP dispose des opérateurs classiques inspirés des langages C et Perl.
  - == égalité
  - < inférieur strict
  - > supérieur strict
  - <= inférieur ou égal
  - >= supérieur ou égal
  - != négation

Logiques

- Les opérateurs logiques sont utilisés dans les tests, par exemple dans un « **if (condition)** »
  - **&&** et
  - || ou
  - **xor** ou exclusif
  - **!** négation
- **Remarque** : les opérateurs **and**, **or** , **not** sont également disponibles et font la même chose.

Arithmétiques

+ addition

- soustraction

/ division

* multiplication

% modulo

++ incrément

-- décrément

Remarque : l'opérateur / renvoie un entier si les 2 opérandes sont des entiers, sinon il renvoie un flottant.

LES OPERATEURS (2)

- **Affectation**
  - = affectation
  - += addition puis affectation
  - -= soustraction puis affectation
  - *= multiplication puis affectation
  - /= division puis affectation
  - %= modulo puis affectation

```
$n = 0;
$n += 2; // $n vaut 2
$n *= 6; // $n vaut 12
$r= $n % 5; // 12 modulo 5 => $r = 2
if( ++$n == 13 ) echo " pas de chance ";
// pré-incrément le test renvoie vrai
```

- **Binaires**
  - & ET
  - | OU
  - ^ XOR
  - ~ NOT

```
echo 3 & 6 ; // 0011 AND 0110 => 2
echo 3 | 6 ; // 0011 OR 0110 => 7
echo 3 ^ 6 ; // 0011 XOR 0110 => 5
echo ~3; // NOT 3 => -4
```

Les structures de contrôles

Syntaxes de **if** :

- If ([condition])
  {
  ...
  }
- If ([condition])
  {
  ...
  } else {
  ...
  }

if (condition)
{ instructions; }
else { instructions; }

Si { instructions; } *ne comporte qu'une instruction, les accolades {} sont optionnels.*

Dans le cas de plusieurs tests successif portant sur une même variable, on utilisera plutôt le test switch.

Syntaxe :

```
switch ( [variable] ) {
        case [valeur1] :
        [bloc d'instructions]
        break;
        case [valeur2] :
        [bloc d'instructions]
        break;
        ...
        default:
        [bloc d'instructions]
                        }
```

La valeur de [variable] est comparé successivement à chaque case. Si il y a égalité, le bloc d'instruction est exécuté.

Il ne faut pas omettre le break en fin de bloc, sans quoi le reste du switch est exécuté.

Enfin, default permet de définir des instructions à effectuer par défaut, c'est à dire si aucun case n'a "fonctionné"...

- **Exemple**

```
switch ($prénom) {
        case "Bob" :
        case "Toto" :
        case "Julien" :
        echo "bonjour ", $prénom , " ! Vous êtes un garçon";
        break;

        case "Anne":
        case "Béatrice" :
        case "Patricia" :
        echo "bonjour ", $prénom , " ! vous êtes une fille";

        default:
        echo "Bonjour $prénom ! Désolé je ne connais pas
        beaucoup de prénoms"
}
```

Les boucles

For (initialisation ; condition ; pas) { instructions; }

Exemple : Affecter une valeur à la variable nbre et afficher la somme des entiers de 1 à nbre.

```
<? Php // Initialisation des variables
$n_nbre = 5;
$n_somme = 0;
for ($i=1; $i<=$n_nbre; $i++)
{ $n_somme = $n_somme + $i; }
echo "La somme des entiers de 1 à $n_nbre est égale à : $n_somme";
?>
```

- En PHP, on dispose des structures de boucle similaires au langage C.
- L'instruction break permet de sortir d'une boucle à tout moment.
- L'instruction continue permet de revenir au début de la boucle.

La boucle WHILE :

while (condition) { instructions; }

Exemple : même exemple

```
<? Php // Initialisation des variables
$n_nbre = 5;
$n_somme = 0;
$i=1;
While($i<=$n_nbre)
{ $n_somme = $n_somme + $i;
$i++; }

echo "La somme des entiers de 1 à $n_nbre est égale à : $n_somme";
?>
```

La boucle DO ... WHILE :

La condition de sortie est située en fin de boucle. Ainsi la boucle est parcourue une fois au minimum.

```
$fp= fopen( "monfichier.txt" );
…
do{
$ligne = fgets( $fp, 1024 );
…}
while( ! feof($fp) );
```

Les tableaux

Déclarations :

```
$fruits= array();
```

Affectations :

```
$fruits[0]= "pomme";
$fruits[1]= "banane";
$fruits[] .= "orange"; // équivaut a $fruits[2]= "orange"
$fruits= array( "pomme", "banane", "orange" );
```

Fonctions :

sizeoff : Renvoie le nombre d'éléments d'un tableau. C'est un équivalent de **count.**

```
$nbelements= sizeof( $tableau );
```

Exemple

- **Initialiser un tableau de 4 cases (contenant des nombres) et en faire la somme.**

```
<?php // Initialisation des variables
$tablo[0]=3; $tablo[1]=2;  $tablo[2]=10; $tablo[3]=5;
$somme = 0;
$i=0;
$nbre = count($tablo);
//parcourt les cases du tableau et effectue la somme
While($i<$nbre)
{ $somme = $somme + $tablo[$i];
  $i++;
}
// affichage de la somme
echo "La somme des nombres du tableau est égale à : $somme";
echo "<BR>";
?>
```

Fonctions (suite):

is_array : renvoie true si la variable est de type tableau (ou tableau associatif), false sinon.

reset : la fonction reset($tableau) place le pointeur interne sur le premier élément du tableau, chaque variable tableau possède un pointeur interne repérant l'élément courant.

end : la fonction end($tableau) place le pointeur interne du tableau sur le dernier élément du tableau.

current : renvoie l'élément courant du tableau.

next : déplace le pointeur vers l'élément suivant, et renvoie cet élément.S'il n'existe pas, la fonction renvoie **false**.

Fonctions (suite):

prev : déplace le pointeur vers l'élément précédent, et renvoie cet élément. S'il n'existe pas, la fonction renvoie **false**.

each : la fonction $a=each($tablo) renvoie l'index et la valeur courante dans un tableau à 2 elements, $a[0] contient l'index, $a[1] la valeur.

list : la fonction list($var1, $varr2, ...) construit un tableau temporaire à partir des variables passées en argument.

key : la fonction key($tablo) renvoie l'index de l'élément courant du tableau.

Fonctions (suite):

sort, rsort, usort, uasort : sont différentes fonctions de tri de tableau.

sort trie par valeurs croissantes, *rsort* par valeurs décroissantes
$tableau_trie = sort($tableau);

usort et *uasor*t permettent au programmeur d'implémenter lui-même la fonction de tri utilisée. PHP appelle successivement la fonction qui doit retourner -1 / 0 / 1 suivant que le premier élément est inférieur / égal / supérieur au second. Dans l'exemple ci-dessous, on implémente un tri qui ne tient pas compte des majuscules/ minuscules

```
function compare_maj( $elem1, $elem2 ) {
if ( strtoupper( $elem1 ) == strtoupper( $elem2 ) ) return 0;
return ( strtoupper( $elem1 ) < strtoupper( $elem2 )) ? -1 : 1; }
……
$tableau_trie = usort( $tableau, "compare_maj" );
```

LES TABLEAUX ASSOCIATIFS

Un tableau associatif est un tableau dont l'index est une chaîne de caractère au lieu d'un nombre. On parle aussi de "hash array" ou "hash". Il se déclare comme un tableau traditionnel, la distinction se fait lors de l'affectation.

Déclarations :

```
$calories= array(); // comme un tableau
```

Affectations :

Affectons un nombre de calories moyen aux fruits.

```
$calories["pommes"]= 300;
$calories["banane"]= 130;
$calories["litchie"]= 30;
```

Fonctions relatives :

isset : pour tester l'existence d'un élément, on utilise la fonction *isset()* .

```
If (isset($calories["pommes"]))
{ echo "une pomme contient ",$calories["pommes"] ,"calories\n"; }
else { echo "pas de calories définies pour la pomme\n"; }
```

asort, arsort, ksort, aksort : Ces fonctions de tri conservent la relation entre l'index et la valeur, généralement le cas dans un tableau associatif.

- ***asort*** trie par valeurs croissantes,
- ***arsort*** par valeurs décroissantes,
- **ksort** trie par index (key) croissantes.

LES FONCTIONS

- A l'image de tout langage structuré, en PHP, une fonction est une suite d'instructions qui peut remplir n'importe qu'elle tâche. Il n'y a pas de distinction fonctions / procédures en PHP.
- Les fonctions PHP prennent de 0 à n paramètres. Ces paramètres peuvent être de type quelconque.
- Remarque : Il faut implémenter la fonction en amont de son utilisation, contrairement au langage C. Dans le cas contraire, PHP sort une erreur du type Call to unsupported or undefined function (fonction) in (file) on line (number).

Déclaration :

La syntaxe de déclaration s'appuie sur le mot clé **function**. Ce mot clé est immédiatement suivi du nom de la fonction par lequel on va l'appeler depuis n'importe quel endroit du code PHP, puis des parenthèses destinées à accueillir les éventuels paramètres.

```
function bonjour() {
 echo " Bonjour ";
 }
bonjour(); // Affiche " Bonjour " à l'écran.
```

Les fonctions peuvent ou non renvoyer un résultat. on utilise l'instruction **return**. La variable retournée peut être de type quelconque. Elle est transmise par copie..

```
function bonjour2() {
 return " Bonjour ";
 }
echo bonjour2() ; // Affiche " Bonjour " à l'écran.
```

Le mode de fonctionnement est sensiblement différent.

Par défaut, les variables globales ne sont pas connues à l'intérieur du corps d'une fonction. On peut cependant y accéder à l'aide du mot-clé **global**.

```
$debug_mode= 1; // variable globale
....
function mafonction()
{
global $debug_mode;
if( $debug_mode )
echo "[DEBUG] in function mafonction()";
....
}
```

Une autre solution est d'utiliser le tableau associatif **$GLOBALS**, qui contient toutes les variables globales déclarées à un instant T :
$GLOBALS['debug_mode'] équivaut à $debug_mode.

Le passage des paramètres par valeur:

Afin de passer des paramètres à la fonction, il suffit de les insérer à l'intérieur des parenthèses prévues à cet effet.

```
function bonjour($prénom, $nom) {
$chaîne = " Bonjour $prénom $nom " ;
// On construit la phrase complète dans la variable locale $chaîne.
return $chaîne ;
// On renvoie la valeur de $chaîne comme résultat de la fonction.
}
.....
echo bonjour("Pierre" , "PAUL") ;
// Affiche " Bonjour Pierre PAUL " à l'écran.
```

Le passage des paramètres par référence :

Par défaut, les paramètres sont transmis par **copie**, c'est à dire que la fonction possède une copie locale de la variable envoyée. Avec la méthode du passage des paramètres par référence, on passe à la fonction l'adresse mémoire d'une variable existante. Cela se fait en précédant de **&** le nom du paramètre. Cela permet de modifier ce paramètre dans la fonction.

```
function bonjour(&$phrase, $prénom, $nom) {
$phrase = " Bonjour $prénom $nom " ;
}
.....
$chaîne = " ";
bonjour($chaîne, "Pierre" , "PAUL") ;
echo $chaîne ; // Affiche " Bonjour Pierre PAUL " à l'écran.
```

Le passage des paramètres par défaut :

Les paramètres optionnels sont autorisés : il suffit de leur affecter une **valeur par défaut**.

```
function mafonction( $param1 = "inconnu", $param2="" ) {
echo "param1=$param1 param2=$param2\n";
}
....
mafonction( "toto", "titi" ); // => "param1=toto param2=titi"
mafonction( "toto" ); // => "param1=toto param2="
mafonction(); // => "param1=inconnu param2="
```

UTILISATION DES TABLEAUX

- Écrire une fonction calculant la factorielle d'un nombre $n donné, à l'aide d'une boucle for.

```
function factorielle( $n )
{
$result =1;
for ( $ i =1; $i<=$n;++$ i ) {
$result *= $i ;
} return $result ;
}
```

```
<?php

// tableau exemple
$tab = array ("1","2","1","3","3","1","4","2","0");

// La fonction array_unique enleve les doublons en
   gardant les clefs
$tab = array_unique ($tab);

// Affichage:
print_r($tab);

?>
```

```php
<?php
 function liste_tableau($tableau)
  {
  while($a=each($tableau))
  {
  echo $a[1] ;
  echo '<br />';
  }
  }

// declarer un tableau, exemple:
  $langage = array("php", "mysql", "pascal", "C");
  // Lister le tableau a l'aide la fonction liste_tableau()
  liste_tableau($langage);
?>
```

```php
<?php

$tab = array("golf", "polo");

if(in_array("golf", $tab)) {
    echo 'La voiture existe !!';
}
// affiche: La variable existe !!

?>
```

LES FICHIERS

PHP fournit plusieurs fonctions qui permettent de prendre en charge l'accès au système de fichiers du système d'exploitation du serveur.

Opérations élémentaires sur les fichiers en PHP :

- **copy($source, $destination)** Copie d'un fichier,
- **$fp=fopen("filemane", $mode)** Ouvre un fichier et retourne un "id" de fichier,
- **fclose($fp)** Ferme un fichier ouvert,
- **rename("ancien", "nouveau")** Renomme un fichier,
- **fwrite($fp, $str)** Ecrit la chaîne de caractères $str,
- **fputs($fp, $str)** Correspond à fwrite(),
- **readfile("filename")** Lit un fichier et retourne son contenu,
- **fgets($fp, $maxlength)** Lit une ligne d'un fichier,
- **fread($fp, $length)** Lit un nombre donné d'octets d'un fichier.

L'accès à un fichierse fait toujours par un **identificateur** de fichier. Cet **"id"** est créé avec la fonction ***fopen()*** et, est requis comme paramètre par la plupart des autres fonctions de fichiers en PHP.

```
$path="/usr/local/apache/htdocs/donnees.txt";
$mode="w";
if ($fp= fopen($path, $mode) ) {
echo "Le fichier a été ouvert";
}
else
echo "Fichier impossible à ouvrir";
if ( close($fp) )
echo " et a été refermé";
?>
```

PROGRAMMATION MODULAIRES

La programmation modulaire permet de la réutilisation de code, notamment par l'écriture de librairies. De ce fait, PHP permet cette modularité par la programmation de librairies classiques et de classes.

Librairies

Les librairies sont des fichiers PHP traditionnels. Leur extension est **.inc** par convention, mais rien n'empêche d'utiliser .PHP. On peut également inclure un fichier HTML ou d'autre type, cependant les éventuels tags PHP ne seront pas interprétés. On inclus un fichier en utilisant les deux instructions **include** ou **require**.

Il existe une différence importante entre les deux :

• Un fichier inclus par ***include*** est inclus dynamiquement, lors de l'exécution du code, c'est-à-dire qu'il est lu puis interprété.

• Un fichier inclus par ***require*** est inclus avant l'interprétation du code. Il est équivalent à la directive ***#include*** du langage C.

On peut comprendre la différence sur l'exemple ci-dessous:

```
if( $user == "Administrateur" ) {
include 'admin_fonctions.inc';
}
if( $user == "Administrateur" ) {
require 'admin_fonctions.inc';
}
```

Avec *include*, le résultat est celui escompté, tandis qu'avec *require*, le fichier *admin_fonctions.inc* est inclus quelque soit le résultat du test if.

- Initialiser un tableau de 4 cases (contenant des nombres) et en faire la somme.

- b) en créant une fonction somme
- c) en créant un fichier spécifique qui contient la fonction somme.

solutions

```
<?php

require ("somme.php");

$tablo[0]=3;
$tablo[1]=2;
$tablo[2]=10;
$tablo[3]=5;

$d_somme = somme ($tablo);

// affichage de la somme
echo "La somme des nombres du tableau est égale à :
    $d_somme";
echo "<BR>";
?>
```

Fonction somme

```
<?php
function somme ($t)
{
$n = count($t); // compte le nombre de cases du tableau
$d_som=0;
$i=0;
While ($i < $n)
{
$d_som=$d_som+$t[$i];
$i=$i+1;
}
return number_format($d_som,2);
}
?>
```

Exercice

- Initialiser un tableau de 4 cases contenant des nombres en Dirhams et en faire la conversion en euros en utilisant une procédure. On affichera la somme totale des cases du tableau en euros ainsi que chaque case du tableau. 1DHS = 8.33 Euro

```php
<?php
require ("fonctions.php");
require ("procedures.php");

$tablo[0]=3; $tablo[1]=2; $tablo[2]=10; $tablo[3]=5;

// mettre le caractère & dans la déclaration de la procédure
// pour transmettre le tableau et non les valeurs du tableau.
conversion ($tablo);
$n_somme = somme ($tablo);
// affichage de la somme
echo "La somme des nombres du tableau est égale à :
    $n_somme";
echo "<BR>";
affichage ($tablo);
?>
```

```php
// CONVERTIT UN TABLEAU DES DIRHAMS  EN EUROS
function conversion (&$t)
{
$d_euro = 8.33;
$i_nbre = count ($t);
$i=0;
While ($i < $i_nbre)
{
$t[$i]=$t[$i]/$d_euro;
$i=$i+1;
}
```

```
// AFFICHE TOUTES LES CASES D'UN TABLEAU EN FORMAT
    NUMERIQUE
function affichage ($t)
{
reset ($t); // se place sur la 1ère ligne du tableau
// parcours toutes les cases du tableau et affecte
//les valeurs des cases et des indices aux 2 variables ind et val
While ((List ($ind , $val) = each($t))== true)
{
echo "la valeur de la case d'indice $ind est égale à " .
    number_format($val,2) .
"<BR>";
}
}
```

Utilisation Fichier texte

$fp=fopen("filemane", $mode)

les paramètres possibles de la fonction fopen() **$mode** :

- r : ouvre en lecture seule, et place le pointeur de fichier au début du fichier.
- r+ : ouvre en lecture et écriture, et place le pointeur de fichier au début du fichier.
- w : ouvre en écriture seule; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
- w+ : ouvre en lecture et écriture; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
- a : ouvre en écriture seule; place le pointeur de fichier à la fin du fichier file. Si le fichier n'existe pas, on tente de le créer.
- a+ : ouvre en lecture et écriture; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.

EXERCICE

- Lire et afficher le contenu d'un fichier texte data.txt
- utiliser les fonctions fopen, fgets, feof, fclose

```php
<?php
$nom = "data.txt";
if (file_exists($nom))
  {
   echo "*** Contenu du fichier $nom : <br>";
   $fichier = fopen($nom, "r"); // ouverture en lecture seule
   while (!feof($fichier))          {
        $une_ligne = fgets($fichier);
        echo "$une_ligne<br>";  }
   echo "*** Fin du contenu  du fichier $nom<br>";
   fclose($fichier);
  }
Else  { echo "Le fichier $nom n'existe pas<br>"; }
?>
```

Ecrire avec la fonction fputs dans ce fichier en remplaçant l'ancien contenu. Vérifier le résultat en exécutant ensuite le script de lecture et affichage.

correction : script

```php
<?php
$nom = "data.txt";
$fichier = fopen($nom, "W"); // ouverture en écriture, l'ancien fichier est détruit
$l1 = "Ce fichier contient cette ligne\n";
$l2 = "et aussi celle ci.\n";
fputs($fichier, $l1);
fputs($fichier, $l2);
fclose($fichier);
?>
```

Ecrire dans ce fichier en ajoutant une ligne au fichier. Vérifier le résultat en exécutant ensuite le script de lecture et affichage.

correction : script

```
<?php
$nom = "data.txt";
$fichier = fopen($nom, "a+"); // ouverture en ajout
$l = "Cette ligne est un ajout.\n";
fputs($fichier, $l);
fclose($fichier);
?>
```

Lire et écrire dans un fichier texte

Réalisation d'un un mini compteur du nombre de visites.

Initialisation

- Tout d'abord créer un fichier compteur.txt
- Placer le chiffre "0" dans ce fichier.

```php
<?php
// on ouvre le fichier compteur.txt en lecture et en écriture.
$fp = fopen ("compteur.txt", "r+");
// on lit le contenu du fichier et on le place dans la variable $nb_visites.
$nb_visites = fgets ($fp, 10);
$nb_visites = $nb_visites + 1;
// on place le pointeur du fichier à l'offset 0 grâce à l'instruction fseek().
fseek ($fp, 0);
// grâce à l'instruction fputs(), on écrit dans notre fichier la nouvelle valeur.
fputs ($fp, $nb_visites);
// on ferme le fichier.
fclose ($fp);
echo 'Ce site compte '.$nb_visites.' visiteurs !';
?>
```

Création des formulaires

```html
<html>
<head>
<title>Ma page de test</title>
</head>
<body>
<form action = "traitement.php" method="post">
Votre nom : <input type = "text" name = "nom"><br />
Votre fonction : <input type = "text" name =
   "fonction"><br />
<input type = "submit" value = "Envoyer">
</form>
</body>
</html>
```

- lorsque l'utilisateur cliquera sur le bouton "Envoyer", les données du formulaire seront envoyées sur la page traitement.php. Et dans la page traitement.php, nous allons récupérer une variable de type tableau ($_POST : car notre formulaire a comme method la valeur post).
- dans la page traitement.php, on aura une variable $_POST['nom'] qui contiendra la chaîne de caractères qu'aura saisi le visiteur dans le champ "Votre nom : " (on a la variable $_POST['nom'], car dans l'attribut name de notre formulaire pour le champ concernant le nom). De même, on aura une variable $_POST['fonction'] qui contiendra la chaîne de caractères qu'aura saisi le visiteur dans la champ "Votre fonction : " (encore une fois, on a la variable $_POST['fonction'] car l'attribut name du champ prend la valeur fonction).

INTRODUCTION AUX BASES DE DONNEES

traitement.php

```
<html>
<head>
<title>Ma page de traitement</title>
</head>
<body>
<?php
// on teste la déclaration de nos variables
if (isset($_POST['nom']) && isset($_POST['fonction'])) {
// on affiche nos résultats
echo 'Votre nom est '.$_POST['nom'].' et votre fonction est
    '.$_POST['fonction'];
}
?>
</body>
</html>
```

- supposons que l'on désire développer une base de données contenant une liste de CD audio. Cette liste de CD sera en fait composée de tous les CD que possède chaque personne d'un groupe d'amis. Et ceci, afin de pouvoir se prêter mutuellement les différents CD, et de savoir exactement qui à quoi comme CD.
- On suppose que le groupe d'amis est composé de x personnes. Chaque personne a un numéro de téléphone, et chaque personne possède un certain nombre de CD. On prendra aussi en considération le titre de l'album et le nom de l'interprète.

- Notez bien que ce tableau, en terme de base de données, se nomme une table et que chaque ligne du tableau se nomme un enregistrement ou un tuple. La première ligne du tableau comporte les attributs de la

Propriétaire	N. tél	Auteur	Titre

- une base de données peut contenir plusieurs tables qu'on peut interroger par des requêtes.
- Faisons maintenant quelques interrogations sur cette base de données :
  - Qui possède l'album y ?
  - Quel est le numéro de téléphone de Monsieur X ?
  - Quels sont les albums Bob Marley disponibles dans la liste de CD ?

- Imaginons maintenant que X vienne de se faire pick-pocketter son tout nouveau portable et qu'il change alors naturellement de numéro.
- Supposons que son nouveau numéro est 061-85-98-78 et qu'en plus il vienne de s'acheter un nouveau CD : Paradis de Bob Sinclar. On insère alors une nouvelle ligne dans notre table.
- Quel est le numéro de téléphone de X ?

ACCES aux SGBD

- L'utilisation en général d'un SGBD (tel que MySQL) avec PHP s'effectue en 5 temps :
  1. Connexion au serveur de données
  2. Sélection de la base de données
  3. Requête
  4. Exploitation des requêtes
  5. Fermeture de la connexion

Connexion au serveur de données

- Pour se connecter au serveur de données, il existe 2 méthodes :
  - Ouverture d'une connexion simple avec la fonction **mysql_connect**
  - Ouverture d'une connexion persistante avec la fonction **mysql_pconnect**
- Remarque : la deuxième méthode diffère de la première par le fait que la connexion reste active après la fin du script.

```
<?
if( mysql_connect("serveur" , $login , $password ) > 0 )
echo "Connexion réussie ! " ;
else
echo "Connexion impossible ! " ;
?>
```

Sélection de la base de données

- Pour faire cette sélection, utilisez la fonction **mysql_select_db** et vous lui passez en paramètre, le nom de votre base.

```
<?
if( mysql_select_db("ma_base" ) == True )
echo "Sélection de la base réussie" ;
else
echo "Sélection de la base impossible" ;
?>
```

- Remarque : les étapes sélection et requête peuvent être faites en même temps, mais il est plus simple surtout pour une seule base, de sélectionner la table avant de commencer les requêtes. Ainsi, toutes les requêtes à venir utiliseront cette base par défaut.

Envoi d'une requête

- Pour envoyer ces requêtes, on peut utiliser 2 fonctions :
  - mysql_query dans le cas où la base de données serait déjà sélectionnée
  - mysql_db_query dans le cas où l'on voudrait sélectionner la base en même temps.

```
<?
  $requête = "SELECT telephone FROM
  liste_proprietaire WHERE nom ="XX"";
  $résultat = mysql_query( $requête );
?>
```

Exploitation des requêtes

- Après l'exécution d'une requête de sélection, les données ne sont pas "affichées", elles sont simplement mises en mémoire, il faut les chercher, enregistrement par enregistrement, et les afficher avec un minimum de traitement.
- PHP gère un pointeur de résultat, c'est celui qui est pointé qui sera retourné. Lorsque vous utilisez une fonction de lecture, le pointeur est déplacé sur l'enregistrement suivant et ainsi de suite jusqu'à la fin.
- Les fonctions qui retournent un enregistrement sont : **mysql_fetch_row**, **mysql_fetch_array** et et prennent comme paramètre l'identifiant de la requête.
- Les 2 exemples suivants partent d'une requête

```
$requête = " SELECT telephone, nom FROM
liste_proprietaire "WHERE nom ="XX"";
$résultat = mysql_query( $requête );
```

mysql_fetch_row : Cette fonction retourne un enregistrement sous la forme d'un tableau simple.

```
<?
    $requete = " SELECT telephone, nom FROM
    liste_proprietaire   WHERE nom ="XX " ";
    $résultat = mysql_query( $requete );

    $enregistrement = mysql_fetch_row($résultat);
    // Affiche le champ - telephone -
echo $enregistrement[0] . "<br>";
// Affiche le champ - nom -
echo $enregistrement[1] . "<br> ";
?>
```

mysql_fetch_array : Cette fonction retourne un enregistrement sous la forme d'un tableau associatif.

```
<?
$enregistrement = mysql_fetch_array ($résultat);
// Affiche le champ - telephone -
echo $enregistrement["telephone"] ."<br>";
// Affiche le champ - nom -
echo $enregistrement["nom"] . "<br>";
?>
```

S'il n'y a pas ou s'il y a plusieurs enregistrements à lire, ces fonctions retournent "false."

Pour savoir combien d'enregistrements ont été retournés par la sélection, la commande **mysql_num_rows** prend comme paramètre l'identifiant de la requête.

```
<?
echo "Il y a " . mysql_num_rows( $résultat ) . " proprietaire ";
while( $enregistrement = mysql_fetch_array( $résultat ))
{
echo $enregistrement['nom'] . " " . $enregistrement[‘telephone'] . "<br>" ;

}
?>
```

Fermeture de la connexion

Vous pouvez fermer la connexion au moyen de la fonction **mysql_close**,

Gestion des erreurs

S'il y a une erreur dans la syntaxe de la requête, on utilise la fonction **mysql_error** qui ne prend pas de paramètres. on écrira un petit message d'erreur si la requête ne se passe pas bien **(or die).**

```
<?
    $résultat = mysql_query( $requête )
    or die ("Erreur dans la requête : " . $requête . "<br>Avec l\"erreur : " .
    mysql_error() );
?>
```

INSERTION DES DONNEES

- Prenons l'exemple de la gestion des CD Audio d'un groupe d'amis. Pour insérer un nouveau propriétaire, il faut fournir au SGBD les informations lui permettant d'insérer un nouvel enregistrement dans la table liste_proprietaire. Ces informations sont :
  - le numéro du nouveau propriétaire
  - le nom du nouveau propriétaire
  - son numéro de téléphone
  - En revanche, comme nous allons le voir, il n'est pas nécessaire de fournir au SGBD le numéro du nouveau propriétaire car cet attribut à été déclaré AUTO_INCREMENT lors de la création de la table. Ceci implique que le SGBD sait, lors d'une nouvelle insertion, qu'il faut prendre le plus grand numéro et qu'il l'augmente de un.
  - On aura alors :

    INSERT INTO liste_proprietaire VALUES ('','XXX','06-98-42-01-36');

```
<?php
// on se connecte à notre base
$base = mysql_connect ('serveur', 'login', 'pass');
mysql_select_db ('ma_base', $base) ;
<html>  <head> <title>Insertion de xx dans la base</title> </head> <body>

// lancement de la requete
$sql = 'INSERT INTO liste_proprietaire VALUES ("", "XX", "06-98-42-01-36")';

// on insère l'enregistrement, on écrira un petit message d'erreur si la
    requête ne se passe pas bien (or die)
mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());
// on ferme la connexion à la base
mysql_close();
?>

</body>  </html>
```

- On désire directement insérer un nouveau propriétaire ainsi qu'un disque lui appartenant. Nous allons voir comment récupérer simplement le nouveau numéro qui vient d'être inséré (donc celui de XX) et ainsi l'utiliser pour insérer notre disque.

```php
<?php
// on se connecte à notre base
$base = mysql_connect ('serveur', 'login', 'pass');
mysql_select_db ('ma_base', $base) ;
// on prépare la requête
$sql = 'INSERT INTO liste_proprietaire VALUES("", "XX", "06-98-42-01-36")';
// on insère le tuple (mysql_query) et au cas où, on écrira un petit message
    d'erreur si la requête ne se passe pas bien (or die)
mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());

// on récupère le dernier numéro inséré, soit le numéro de XX
$numero_insere = mysql_insert_id();
// on insère le tuple (mysql_query) et au cas où, on écrira un petit
    message d'erreur si la requête ne se passe pas bien (or die)
$sql = 'INSERT INTO liste_disque VALUES ("'.$numero_insere.'",
    "SOPRANO", "HALA HALA")';
// on insère le tuple (mysql_query) et au cas où, on écrira un petit
    message d'erreur si la requête ne se passe pas bien (or die)
mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());
// on ferme la connexion à la base
mysql_close();
?>
```

INSERTION DYNAMIQUE

- On veut rendre nos insertions dynamiques en effectuant tout simplement nos insertions à partir des valeurs fournies par un formulaire.
- Imaginons que l'on désire insérer des nouveaux disques. Supposons que l'on dispose d'une page html contenant un formulaire permettant de saisir le nom du propriétaire, et que ce formulaire vous demande également le titre d'un album ainsi que son interprète (on suppose également que le champ action de notre formulaire correspond au nom de la page PHP qui traite les données, soit la page contenant le code ci-dessous).
- On suppose enfin, que le champ du formulaire contenant le nom du propriétaire porte le nom proprio (on pourra alors utiliser la variable $_POST['proprio'] dans notre page PHP, tout en supposant de notre formulaire à une méthode POST), que le champ contenant l'interprète porte le nom interprete et que le champ contenant le titre porte le nom titre.

On aura alors :

```
<?php
// on se connecte à notre base
$base = mysql_connect ('serveur', 'login', 'pass');
mysql_select_db ('ma_base', $base) ;
?>
<html>
<head>
<title>Insertion de nouveaux disques dans la base</title>
</head>
<body>
<?php
// on teste si les variables du formulaire sont bien déclarées
if (isset($_POST['proprio']) && isset($_POST['interprete']) && isset($_POST['titre']))
    {
// on prépare la requête pour récupérer le numero du propriétaire
$sql = 'SELECT numero FROM liste_proprietaire WHERE nom =
    "'.$_POST['proprio'].'"' ;
```

```
// on lance la requête (mysql_query) et on impose un message d'erreur si la requête ne se passe pas bien (or die)
$req = mysql_query($sql) or die('Erreur SQL !<br />'.$sql.'<br />'.mysql_error());
// on récupère le résultat sous forme d'un tableau
$data = mysql_fetch_array($req);
// on libère l'espace mémoire alloué pour cette interrogation de la base
mysql_free_result ($req);
// on insère le tuple (mysql_query) et au cas où, on écrira un petit message d'erreur si la requête ne se passe pas bien (or die)
$sql = 'INSERT INTO liste_disque VALUES("'.$data['numero'].'", "'.$_POST['interprete'].'", "'.$_POST['titre'].'")';
// on insère le tuple (mysql_query) et au cas où, on écrira un petit message d'erreur si la requête ne se passe pas bien (or die)
mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());
// on ferme la connexion à la base
mysql_close();

echo 'Nous venons d\'insérer un nouveau disque : '.$_POST['titre'].' de '.$_POST['interprete'].' appartenant à '.$_POST['proprio'];
}
else {
echo 'Les variables du formulaire ne sont pas déclarées';
}
?>
</body>
</html>
```